LARAVEL

Manual Book for Web Developers

By

Tin Mai Zaw

2025 EDITION

Northern City, Yangon, Myanmar

©2025 Northern City. All Rights Reserved

အမှာစကား

ဤစာအုပ်သည် Laravel ကို basic lavel ကနေ advanced level အထိ လေ့လာလို့ရအောင် reading အပိုင်းနဲ့ coding အပိုင်းတွေကို အသေးစိတ် ရှင်းလင်းချက်တွေနဲ့ စုံစုံလင်လင် ရေးသားပြုစုထားတဲ့ လက်စွဲစာအုပ်ဖြစ်ပါတယ်။ Northern City Laravel လက်စွဲစာအုပ် ကို 2018 တုန်းက English လို ထုတ်ဝေခဲ့တာ ဖြစ်ပါတယ်။ အခု ဒုတိယအကြိမ် 2025 April လမှာ မြန်မာလို ဒုတိယ အကြိမ် ထုတ်ဝေခြင်းဖြစ်ပါတယ်။ Laravel Reference Books, Online Laravel Reference Website တွေအပြင် Chat GPT, Deepseek AI tools တွေသုံးပြီး အချို့သင်ခန်းစာတွေကို ထပ်မံ ဖြည့်စွက်ထားပါတယ်။

မြန်မာလိုထုတ်ဝေရခြင်း အဓိက အကြောင်းအရင်းကတော့ Northern City Web Application ကျောင်းသူကျောင်းသားဟောင်းများ ၊ ကျောင်းသားသစ်များအားလုံးကို ထိထိရောက်ရောက် ကူညီနိုင်အောင်လို့ ဖြစ်ပါတယ်။ ခိုင်မာတဲ့ ဘဝရပ်တည်ချက်တွေ တည်ဆောက်နိုင်ဖို့ ၊ Laravel နည်းပညာနဲ့ပတ်သက်လို့ အရည်အသွေး စွမ်းဆောင်ရည်တွေထက်မြက်နေစေဖို့ ရည်ရွယ်ပါတယ်။ Laravel subject နဲ့ ပါတ်သက်လို့ လေ့ကျင့်ခန်းတွေကို ခရေစေ့တွင်းကျ နားလည်အောင် ဖတ်ဖို့လိုမယ်။ လေ့ကျင့်နေဖို့လည်းလိုပါတယ်။ ပြီးတော့ ကွန်ပျူတာကျောင်း သူကျောင်းသားများ၊ အင်ဂျင်နီယာအိုင်တီ ကျောင်းသူကျောင်းသားများကို တစ်ဖက်တစ်လမ်း အထောက်အကူပြုစေရန်အတွက် ရည်ရွယ် ထုတ်ဝေရခြင်းလည်းဖြစ်ပါတယ်။

ဤစာအုပ်ကို မလေ့လာခင်မှာ HTML, CSS, JavaScript တွေကို လေ့လာထားဖို့တော့ လိုပါမယ်။ Web development ကို စတင်လေ့လာနေတဲ့ သူတွေအတွက် မဟုတ်ပါဘူး။ Web design သို့မဟုတ် web frontend development နဲ့ ပါတ်သက်လို့ အတွေ့အကြုံရှိပြီးသား သူတွေအတွက် ပိုပြီးသင့်တော်ပါတယ်။ တကယ်လို့ HTML, CSS တွေမလေ့လာရသေးဘူးဆိုရင်တော့ ကျနော်တို့ Northern City Computer Training Cetner ရဲ့ Youtube Channel နဲ့ Facebook Page တွေမှာ သွားရောက် ပြီး Programming Basic, HTML, CSS ,JavaScript tutorial တွေကို အခမဲ့ လေ့လာနိုင်ပါတယ်။ Northern City Youtube Channel Link နဲ့ Facebook Page Link တွေကတော့ အောက်ပါအတိုင်းဖြစ်ပါတယ်။

Youtube Channel:

https://www.youtube.com/@northerncity2022

Facebook Page:

https://www.facebook.com/northerncity.programmingspace

စာတွေကြီးပဲ ဖတ်နေတာမျိုးမဟုတ်ပဲ hand-on practice လက်တွေ့သင်ခန်းစာတွေကို source code တွေနဲ့ ရေးသားဖော်ပြထားပါတယ်။ မကြာခင်မှာ ဒီစာအုပ်ရဲ့ လက်တွေ့ လေ့ကျင့်ခန်း တွေကို video tutorial နဲ့ github repository file တွေနဲ့ ရောပြီး ပြန်လည် ထုတ်ဝေသွားမှာ ဖြစ်ပါတယ်။

Web Developer အနေနဲ့ အသက်မွေးဝမ်းကျောင်းပြုမယ်ဆိုရင် အောက်ပါအတိုင်း web developer career path ကို အဆင့်ဆင့်လေ့လာဖို့ အကြံပြုပါတယ်။ လေ့လာရမယ့် ဘာသာရပ်နဲ့ ခန့်မှန်းကြာချိန်တွေရေးပေးထားပါတယ်။ Intensive study duration ကို ခန့်မှန်းဖော်ပြခြင်းဖြစ်ပါတယ်။ အားတဲ့အချိန်မှ ၊ စိတ်ပါတဲ့အချိန်မှ လေ့လာမယ်ဆိုတဲ့ ပုံစံ လုံးဝမဟုတ်ပါဘူး။ Popular ဖြစ်တဲ့ web application ဘာသာရပ်တွေကိုပဲ ရေးသားဖော်ပြ ထားခြင်း ဖြစ်ပါတယ်။

✓ Suggested Web Developer Career Path

1. UIUX
   - ✓ Photoshop, Illustrator, Figma, Adobe XD
   - 🛈 အနည်းဆုံး ၂ လ အချိန်ပေးသင့်ပါတယ်။

2. Frontend Web Design
   - ✓ HTML, CSS, JavaScript, Wordpress, Bootstrap, Tailwind CSS, Sass, Git & Github
   - 🛈 အနည်းဆုံး ၄ လ လောက် အချိန် ပေးသင့်ပါတယ်။

3. Web Backend
   - ✓ Php, Laravel, Node JS, Django, Flask, Springboot, C#.NET Core
   - ✓ MySQL, SQLite, PostgreSQL, AWS, Mongo DB, Redis , Firebase , Cassandra DB
   - ✓ Github, BitBucket, Docker
   - 🛈 အနည်းဆုံး ၆ လ လောက် အချိန် ပေးသင့်ပါတယ်။

4. Web API
   - ✓ Vue JS, React JS, Angular JS, Next JS
   - ✓ MySQL, SQLite, PostgreSQL, AWS, MongoDB, Redis, Firebase, Cassandra DB
   - ✓ Postman, Insomnia
   - ✓ Github , BitBucket, Docker
   - 🛈 အနည်းဆုံး ၆ လ လောက် အချိန် ပေးသင့်ပါတယ်။

မာတိကာ

1.	Laravel သမိုင်းအကျဉ်း	စာမျက်နှာ	၅
2.	Laravel ကို ဘာလို့ လေ့လာသင့်သလဲ	စာမျက်နှာ	၈
3.	Hardware & Software Requirements	စာမျက်နှာ	၁၈
4.	Project Installation	စာမျက်နှာ	၂၂
5.	Project Files	စာမျက်နှာ	၂၈
6.	MVC	စာမျက်နှာ	၃၂
7.	Blade Template Engine	စာမျက်နှာ	၄၇
8.	Route	စာမျက်နှာ	၅၃
9.	CRUD	စာမျက်နှာ	၅၉
10.	Upload File	စာမျက်နှာ	၇၀
11.	Model Relationships	စာမျက်နှာ	၈၀
12.	Ajax	စာမျက်နှာ	၁၀၂
13.	Database Seeding	စာမျက်နှာ	၁၂၀
14.	Data Table	စာမျက်နှာ	၁၃၄
15.	Shopping Cart	စာမျက်နှာ	၁၄၆
16.	Authentication	စာမျက်နှာ	၁၆၅
17.	Middleware	စာမျက်နှာ	၁၈၇
18.	Roles & Permissions	စာမျက်နှာ	၁၉၇
19.	Sending Email	စာမျက်နှာ	၂၃၉
20.	Export Documents	စာမျက်နှာ	၂၅၃
21.	Payment Gateway	စာမျက်နှာ	၂၆၅

22. OTP	စာမှက်နှာ	၂၉၁
23. Two Factor Authentication	စာမှက်နှာ	၃၀၄
24. Chat	စာမှက်နှာ	၃၁၁
25. Comments System	စာမှက်နှာ	၃၂၉
26. API	စာမှက်နှာ	၃၄၂
27. Livewire	စာမှက်နှာ	၃၇၅
28. Git & Github	စာမှက်နှာ	၃၉၃
29. Docker	စာမှက်နှာ	၄၀၄
30. Web Deployment	စာမှက်နှာ	၄၁၅
31. How to land a Job	စာမှက်နှာ	၄၂၀
32. References	စာမှက်နှာ	၄၂၇
33. စာရေးသူသမိုင်းအကျဉ်း	စာမှက်နှာ	၄၂၈
34. စာရေးသူအိုင်တီးအတွေ့အကြုံမှတ်တမ်း	စာမှက်နှာ	၄၂၉
35. စာရေးသူ Documentary Photos	စာမှက်နှာ	၄၃၀

Laravel History

သမိုင်းအကျဉ်း

Laravel သည် PHP ဖြင့် ရေးသားထားတဲ့ Open-Source Web Application Framework တစ်ခု ဖြစ်ပါတယ်။ ၎င်းကို Taylor Otwell က 2011 ခုနှစ်မှာ စတင် ဖန်တီးခဲ့ပါတယ်။ Laravel ရဲ့ ရည်ရွယ်ချက်မှာ Developer တွေအတွက် Elegant, Simple နဲ့ Readable Code ရေးနိုင်အောင် ကူညီပေးဖို့ ဖြစ်ပါတယ်။

အစောပိုင်း ဗားရှင်း (2011–2013) တွေကတော့ Laravel 1 (June 2011) – ပထမဆုံး Version အဖြစ် Release လုပ်ခဲ့ပါတယ်။ ဒီ Version မှာ Basic Features တွေ ပါဝင်ခဲ့ပေမယ့် Database ORM (Object-Relational Mapping) မပါသေးပါဘူး။ Laravel 2 (September 2011) – ဒီ Version မှာ Controllers, Views နဲ့ Templating Engine (Blade) တို့ ထည့်သွင်းပေးခဲ့ပါတယ်။ Laravel 3 (February 2012) – Command-Line Tool (Artisan), Database Migrations, Eloquent ORM နဲ့ Unit Testing Support တို့ ပါလာပါတယ်။

2013 ခုနှစ်မှာ Laravel 4 (Illuminate Components) ကို Release လုပ်ခဲ့ပါတယ်။ ဒီ Version မှာ Composer (PHP Dependency Manager) ကို အသုံးပြုပြီး Modular System ဖြစ်လာခဲ့ပါတယ်။ ဒါ့အပြင် Queues, Mail Sending, Event Handling စတဲ့ Features တွေ ထည့်သွင်းပေးခဲ့ပါတယ်။

Laravel 5 ကို 2015 ခုနှစ်မှာ မိတ်ဆက်ခဲ့ပါတယ်။ ဒီ Version မှာ အသစ်ဖြစ်တဲ့ Features တွေ ပါဝင်လာပါတယ်။ Blade Improvements – Template Inheritance နဲ့ Components တို့ကို ပိုကောင်းအောင် ပြုလုပ်ထားပါတယ်။ Scheduler – Task Scheduling လုပ်နိုင်အောင် ထည့်သွင်းပေးထားပါတယ်။ Authentication Scaffolding (make:auth) – Login/Register System

ကို အလွယ်တကူ Generate လုပ်နိုင်ပါတယ်။ Laravel Elixir (Frontend Asset Management) – Webpack (Mix) မတိုင်ခင် အသုံးပြုခဲ့တဲ့ Tool ဖြစ်ပါတယ်။

နောက်ဆုံး Version တွေဖြစ်တဲ့ ၂၀၁၇ ကနေ ဒီနေ့အထိ version တွေမှာ အပြောင်းအလဲ တွေနဲ့ တိုးတက်မှုတွေများကြီး ရှိပါတယ်။ Laravel 5.5 (LTS – Long Term Support, 2017) – Auto-Discovery, API Resources, Error Handling Improvements တို့ ပါဝင်လာပါတယ်။ Laravel 6 (2019) – Semantic Versioning ကို စတင် အသုံးပြုခဲ့ပါတယ်။ Laravel Vapor (Serverless Deployment) နဲ့ Job Middleware တို့ မိတ်ဆက်ခဲ့ပါတယ်။ Laravel 7 (2020) – Custom Eloquent Casts, HTTP Client, Fluent String Operations တို့ ပါဝင်လာပါတယ်။ Laravel 8 (2020) – Jetstream (Authentication Scaffolding), Laravel Fortify, Model Factories Improvement တို့ ပါဝင်လာပါတယ်။ Laravel 9 (2022) – Symfony 6, Flysystem 3, New Query Builder Interface တို့ ပါဝင်လာပါတယ်။ Laravel 10 (2023) – Process Isolation, Laravel Pennant (Feature Flags), Invokable Validation Rules တို့ ပါဝင်လာပါတယ်။

Laravel မှာ အခြား Powerful Tools တွေ ပါဝင်ပါတယ်။ အဲဒါတွေကတော့ Laravel Forge – Server Management Tool ၊ Laravel Vapor – Serverless Deployment Platform ၊ Laravel Nova – Admin Panel နဲ့ Laravel Echo – Real-Time Event Broadcasting တွေဖြစ်ပါတယ်။

Laravel သည် PHP Developer တွေအတွက် အထူးကောင်းမွန်တဲ့ Framework တစ်ခု ဖြစ်ပါတယ်။ လွယ်ကူမြန်ဆန်တဲ့ Development Process, Clean Code Structure, နဲ့ ကြီးမားတဲ့ Community Support တို့ကြောင့် လူကြိုက်များလာခဲ့ပါတယ်။ ဒါကြောင့် Laravel ကို 2024 အထိ ဆက်လက် အသုံးပြုနေကြပြီး အနာဂတ်မှာလည်း ပိုမိုကောင်းမွန်တဲ့ Features တွေ ထပ်မံ ထွက်ပေါ်လာဦးမယ်လို့ မျှော်လင့်ရပါတယ်။

Laravel ကို ဘာလို့ သုံးသင့်သလဲ?

Laravel ကို ဘာလို့ သုံးသင့်သလဲ?

Laravel သည် PHP ဖြင့် ရေးသားထားတဲ့ **Modern Web Application Framework** တစ်ခု ဖြစ်ပြီး၊ Developer တွေအတွက် လွယ်ကူမြန်ဆန်စွာ Web App တွေ ဖန်တီးနိုင်အောင် ကူညီပေးပါတယ်။

☑ Laravel သုံးသင့်တဲ့ အဓိက အကြောင်းရင်းများ

✓ Web Hosting Price

ကျောင်းတုန်းက ဆရာ/ဆရာမ တွေ Java ကအရမ်းကောင်းတယ်။ Python က အရမ်းကောင်းတယ်။ Software company တော်တော်များများ JavaScript နဲ့ Python ကို ပြောင်းသုံးနေကြပြီ။ ဒါပေမယ့် ကျနော် ဘွဲ့လဲ ရပြီး Java နဲ့ web application တွေရေးတယ်။ Online ပေါ် project deployment လုပ်မယ် လုပ်တော့ ဇာတ်လမ်းကစပြီ။ Hosting price က အရမ်းစျေးကြီးပါတယ်။ ဘယ်လိုမှ မတတ်နိုင်ဘူး။ Python နဲ့ JavaScript project တွေလည်း ဒီအတိုင်းပဲ။ Application Project တွေရဲ့ ကုန်ကျစရိတ်ကို တွက်ကြည့်ပြီး ဒီတိုင်းရပ်ထားရတော့တယ်။ ကိုယ်က beginner လည်းဖြစ်တယ်။ Budget လည်း ကိုယ့်မှာ အလုံအလောက်မတတ်နိုင်ဘူး ဆိုရင် အကောင်းဆုံး ရွေးရမည့် web programming language ကတော့ Laravel ပဲ ဖြစ်ပါတယ်။ Affordable, Cheap and secure ဖြစ်တာမို့ beginners တွေအတွေက်ရော freelancers တွေအတွက် အသင့်တော်ဆုံး

language ဖြစ်ပါတယ်။ Web Language တွေ ရဲ့ Hosting Price တွေအောက်မှာ နှိုင်းယှဉ် ဖော်ပြပေးထားပါတယ်။

✓ PHP/Laravel Web Hosting

💵 စျေးနှုန်း:

- Shared Hosting: တစ်လကို US dollar 1 dollar ကနေ 10 dollar ကြား (အခြေခံ WordPress, Joomla အတွက်)
- VPS/Cloud: တစ်လကို US dollar 5 ကနေ 50 dollar ကြား (PHP-FPM, Laravel အတွက်)

✅ အားသာချက်များ:

- စျေးသက်သာ (Shared Hosting တွင် အလွယ်တကူရနိုင်)
- လွယ်ကူသော Setup (cPanel, Softaculous ဖြင့် 1-click install)
- မြန်ဆန်သော Performance (OPCache, JIT Compiler in PHP 8+)
- ကျယ်ပြန့်သော CMS Support (WordPress, Drupal, Magento)
- MySQL/MariaDB နှင့် အဆင်ပြေ Integration

❌ အားနည်းချက်များ:

- Scalability အားနည်း (Enterprise-level apps အတွက် မသင့်တော်)
- Shared Hosting တွင် Security Risks များ (Poorly coded scripts များကြောင့်)
- Multithreading မပါဝင် (Java, Node.js လောက် မမြန်ဆန်)

✓ Java Web Hosting

💵 စျေးနှုန်း:

- VPS/Dedicated Server: တစ်လကို US dollar 20 dollar ကနေ 500 ကြား (Tomcat, Spring Boot အတွက်)
- Cloud (AWS, GCP): Pay-as-you-go (ပိုမိုစျေးကြီး)

✅ အားသာချက်များ:

- မြင့်မားသော Performance (JVM ကြောင့် စွမ်းဆောင်ရည်ကောင်း)
- Enterprise-ready (Banking, Large-scale apps အတွက် သင့်တော်)
- Multithreading Support
- Spring Framework နှင့် Strong Ecosystem

❌ အားနည်းချက်များ:

- စျေးကြီး (Shared Hosting မရှိသလောက် ဖြစ်သည်)
- Memory Usage များ (JVM ကြောင့် RAM လိုအပ်ချက်မြင့်)
- Complex Deployment (Tomcat, JBoss စသည့် Server Config လိုအပ်)

✓ Python Web Hosting

💵 စျေးနှုန်း:

- Shared Hosting: Rare (အများအားဖြင့် VPS/Cloud တွင်သာ)
- VPS/Cloud: တစ်လကို 10 USD ကနေ 100 USD ကြား (Django, Flask အတွက်)

✅ အားသာချက်များ:

- Developer-friendly Syntax (ရေးရလွယ်၊ ဖတ်ရလွယ်)
- Data Science & AI Integration (TensorFlow, PyTorch)
- Django/Flask Frameworks များ ကောင်းမွန်
- Linux/Unix တွင် Native Support

❌ အားနည်းချက်များ:

- Shared Hosting မရှိသလောက် (အများစုမှာ VPS လိုအပ်)
- GIL (Global Interpreter Lock) ကြောင့် Performance အားနည်း (Multithreading ပြဿနာ)
- Node.js/Java ထက် နှေးနိုင်သည်

✓ Node.js Web Hosting

💵 ဈေးနှုန်း:

- Shared Hosting: Rare (အများအားဖြင့် VPS/Cloud)
- VPS/Cloud: တစ်လကို 5 USD ကနေ 100 USD ကြား (PM2, Nginx Reverse Proxy လိုအပ်)

✅ အားသာချက်များ:

- မြန်ဆန်သော Performance (Non-blocking I/O ကြောင့်)
- Real-time Apps အတွက် ကောင်း (Chat, Gaming, Streaming)
- JavaScript Full Stack (Frontend + Backend တစ်ပြိုင်နက်)
- npm Ecosystem ကြီးမား

❌ အားနည်းချက်များ:

- Single-threaded (CPU-intensive tasks အတွက် မကောင်း)
- Callback Hell ပြဿနာ (Async/Await ဖြင့် လျော့ချနိုင်)
- Shared Hosting မရှိခြင်း

✓ C# (ASP.NET) Web Hosting

💵 ဈေးနှုန်း:

- Windows Server Hosting: တစ်လကို 10USD ကနေ 200 USD ကြား (IIS လိုအပ်)
- Cloud (Azure): Pay-as-you-go (Microsoft Ecosystem တွင် စျေးကြီး)

✅ အားသာချက်များ:

- Microsoft Ecosystem နှင့် အဆင်ပြေ (SQL Server, Azure)
- High Performance (Java နှင့် နှိုင်းယှဉ်နိုင်)
- Visual Studio IDE နှင့် Strong Tooling
- Enterprise Apps အတွက် ကောင်း

❌ အားနည်းချက်များ:

- Windows Server လိုအပ်ခြင်း (Linux တွင် .NET Core သုံးနိုင်သော်လည်း အချို့ Features မရ)
- PHP/Node.js ထက် စျေးကြီး
- Open Source Ecosystem အားနည်း (Java, Python နှင့်ယှဉ်လျှင်)

📊 နှိုင်းယှဉ်ချက် Summary

Language	Monthly Price (USD)	Best For	Performance	Scalability	Ease of Use
PHP/Laravel	1–10	Small Websites, CMS	⭐⭐⭐	⭐⭐	⭐⭐⭐⭐
Java	20–200+	Enterprise Apps	⭐⭐⭐⭐⭐	⭐⭐⭐⭐	⭐⭐
Python	10–100	AI, Data Apps	⭐⭐⭐	⭐⭐⭐	⭐⭐⭐⭐
Node.js	5–100	Real-time Apps	⭐⭐⭐⭐	⭐⭐⭐	⭐⭐⭐
C#	10–200	Microsoft Apps	⭐⭐⭐⭐	⭐⭐⭐⭐	⭐⭐

Web Language ရွေးချယ်ရန် အကြံပြုချက်:

- စျေးသက်သာပြီး ရိုးရှင်းသော ဝဘ်ဆိုဒ် → PHP/Laravel
- အကြီးစား Enterprise System → Java / C#
- Real-time Apps (Chat, Streaming) → Node.js
- Data Science / AI → Python
- Microsoft Products (SQL Server, Azure) → C#

✓ Elegant Syntax & Clean Code Structure

Laravel မှာ **ရှင်းလင်းပြီး လှပတဲ့ Code Structure** ရှိပါတယ်။ ဥပမာ -

Code:

```
// Route ကို ရိုးရှင်းစွာ သတ်မှတ်နိုင်ခြင်း
Route::get('/home', [HomeController::class, 'index']);
```

ဒီလို Syntax တွေကို အသုံးပြုရတာ လွယ်ကူပြီး နားလည်ရလွယ်ပါတယ်။

✓ Built-in Tools & Features (အဆင်သင့် Feature များ)

Laravel မှာ အခြား Framework တွေထက် Built-in Features အများကြီး ပါဝင်ပါတယ်။ Authentication System (php artisan make:auth) – Login, Register စနစ်ကို မိနစ်ပိုင်းအတွင်း တည်ဆောက်နိုင်ပါတယ်။ Eloquent ORM – Database နဲ့ ချိတ်ဆက်ရာမှာ SQL Query တွေ ရိုးရိုးရှေးစရာ မလိုဘဲ PHP Code နဲ့ အလွယ်တကူ အသုံးပြုနိုင်ပါတယ်။ Artisan CLI – Command

Line Tool ဖြင့် Code Generate လုပ်ခြင်း၊ Database Migration လုပ်ခြင်း စတာတွေကို အလွယ်တကူ လုပ်ဆောင်နိုင်ပါတယ်။

✓ Security (လုံခြုံရေး ကောင်းမွန်ခြင်း)

Laravel မှာ အားနည်းချက်တွေကို ကာကွယ်ပေးတဲ့ Built-in Security Features တွေ ပါဝင်ပါတယ်။ CSRF Protection – Form Submission တွေမှာ Cross-Site Request Forgery အန္တရာယ်ကို ကာကွယ်ပေးပါတယ်။ SQL Injection Prevention – Eloquent ORM က Parameter Binding ကို အလိုအလျောက် လုပ်ဆောင်ပေးပါတယ်။ Password Hashing – User Password တွေကို Bcrypt နဲ့ Auto-Hash လုပ်ပေးပါတယ်။

✓ Database & Migration Support

Laravel မှာ Database Migration & Seeding စနစ်ပါဝင်ပြီး၊ Team Project တွေမှာ Database Structure ကို Version Control လုပ်နိုင်ပါတယ်။

Code:

```
// Migration Example
php artisan make:migration create_users_table
```

ဒါ့အပြင် Multiple Database Connections (MySQL, PostgreSQL, SQLite, SQL Server) ကို Support လုပ်ပါတယ်။

✓ Blade Templating Engine

Laravel ရဲ့ Blade Template Engine က PHP ထဲမှာ HTML ကို Clean ဖြစ်အောင် ရေးနိုင်ပါတယ်။

Code:

```
<!-- Blade Syntax Example -->
@if($user->isAdmin())
```

```
    <p>Welcome Admin!</p>
@endif
```

Blade က **Fast & Lightweight** ဖြစ်ပြီး၊ ရှုပ်ထွေးတဲ့ PHP Code တွေကို ရှင်းလင်းစွာ ဖော်ပြနိုင်ပါတယ်။

✓ Testing & Debugging (စမ်းသပ်ရလွယ်ခြင်း)

Laravel မှာ **PHPUnit** နဲ့ Integration လုပ်ထားပြီး Unit Testing လုပ်ရတာ အရမ်းလွယ်ပါတယ်။

Code:

```
// Test Example
public function test_login_page()
{
    $response = $this->get('/login');
    $response->assertStatus(200);
}
```

ဒါ့အပြင် Telescope & Debugbar လို Tools တွေနဲ့ Error တွေကို အလွယ်တကူ Debug လုပ်နိုင်ပါတယ်။

✓ Community & Ecosystem (ကြီးမားတဲ့ Community)

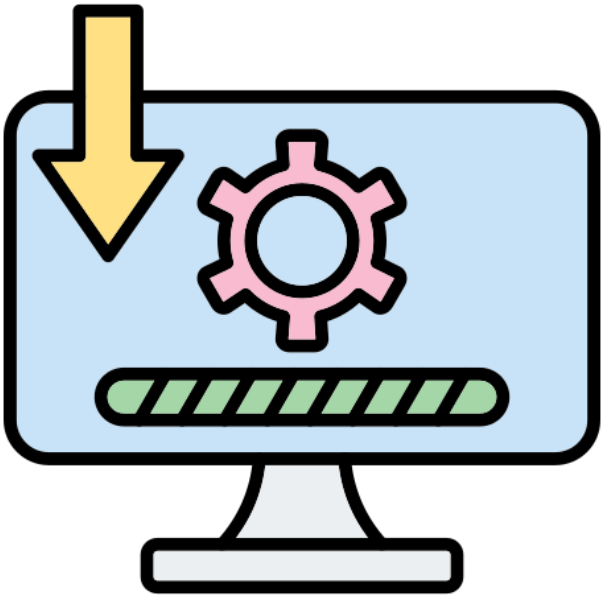
Laravel မှာ ကမ္ဘာတစ်ဝှမ်းက Developer တွေရဲ့အကူအညီ ရနိုင်ပါတယ်။ Laracasts Video Tutorials တွေအများကြီး ရှိနေတယ်။ updated Laravel News တွေ Laravel Article တွေ အမြဲတယ်း ဖတ်လို့ရတယ်။ GitHub, Stack Overflow, Discord Groups တွေမှာ အလွယ်တကူ အကူအညီ တောင်းလို့ရတယ်။

✓ API Development (RESTful API ဖန်တီးရလွယ်ခြင်း)

Laravel က API Development အတွက် အထူးကောင်းမွန်ပါတယ်။ API Resource – JSON Data ကို အလွယ်တကူ Format လုပ်နိုင်ပါတယ်။ Sanctum / Passport – API Authentication (OAuth, Token-based) အတွက် Built-in Packages တွေ ပါဝင်ပါတယ်။

Laravel ကို သုံးသင့်တဲ့ အဓိက အကြောင်းရင်းမှာ လွယ်ကူမြန်ဆန်တဲ့ Development Process, လုံခြုံမှု, Scalability နဲ့ ကြီးမားတဲ့ Community Support တို့ကြောင့် ဖြစ်ပါတယ်။ အကယ်၍ သင်ဟာ စတင်လေ့လာနေတဲ့ Developer တစ်ယောက် ဖြစ်မယ် ဆိုရင် Laravel က စတင်ဖို့ အကောင်းဆုံး Framework တစ်ခု ဖြစ်ပါတယ်။ Professional Project တွေမှာ လည်း Laravel ကို ယုံကြည်စွာ အသုံးပြုနိုင်ပါတယ်။

SYSTEM REQUIREMENTS & SOFTWARE TOOLS



System Requirements & Software Tools

Laravel Application လေ့ကျင့်ခန်းတွေကို မလုပ်ခင်မှာ Laravel application development အတွက် လိုအပ်သော computer hardware နှင့် software requirements များကို အသေးစိတ် ဖော်ပြပေးမယ်။

---

1. Hardware Requirements (ကွန်ပျူတာ အခြေခံလိုအပ်ချက်များ)

Laravel ကို ချောမွေ့စွာ အလုပ်လုပ်နိုင်ဖို့ အနည်းဆုံး အောက်ပါ hardware specifications တွေ လိုအပ်ပါတယ်။

✓ Minimum Requirements (အနည်းဆုံး)

- Processor (CPU): Dual-core 1.5GHz (Intel/AMD)
- RAM: 4GB (8GB ဆိုပိုကောင်း)
- Storage (HDD/SSD): 10GB free space (SSD ဆိုပိုမြန်မယ်)
- Operating System:
    - Windows 10/11 (64-bit)
    - macOS (Latest)
    - Linux (Ubuntu, Fedora, CentOS)

✓ Recommended Hardware Requirements (အကောင်းဆုံး Performance အတွက်)

- Processor (CPU): Quad-core 2.5GHz+ (Intel i5/i7, AMD Ryzen)
- RAM: 8GB+ (16GB ဆိုပိုကောင်း)
- Storage: 256GB SSD (Database, Cache အတွက် မြန်ဆန်မှု လိုအပ်ပါတယ်)
- Graphics: Integrated GPU နဲ့ အဆင်ပြေပေမယ့် Frontend (Vue/React) သုံးမယ်ဆိုရင် Dedicated GPU ကောင်းကောင်း လိုမယ်။

---

2. Software Requirements (လိုအပ်သော Software များ)

✓ Operating System (OS) ရွေးချယ်မှု

Laravel ကို Windows, macOS, Linux တို့မှာ Run နိုင်ပါတယ်။

- Windows: WSL2 (Windows Subsystem for Linux) သုံးရင် ပိုကောင်းပါတယ်။
- macOS: Native Support ရှိပါတယ်။
- Linux: Ubuntu/Debian က Laravel development အတွက် အကောင်းဆုံး။

✓ Web Server & PHP

- PHP 8.1+ (Laravel 12 အတွက်)
- Web Server:
  - Apache (with `mod_rewrite`)
  - Nginx (Recommended for Production)
- Database:
  - MySQL 5.7+ / MariaDB
  - PostgreSQL / SQLite / SQL Server

✓ Development Tools (IDE & Terminal)

- Code Editors:
  - VS Code (Recommended) + Extensions (PHP Intelephense, Laravel Artisan, Blade)
  - PHPStorm (Paid version က Laravel အတွက် ကောင်းဆုံးဖြစ်ပါတယ်။)
- Terminal:
  - Windows: Git Bash / CMDer
  - macOS/Linux: Terminal / iTerm2

✓ Dependency Management & Build Tools

- Composer (PHP Package Manager) → Laravel Install လုပ်ဖို့
- Node.js (v16+) + NPM/Yarn → Frontend (Vue, React) နဲ့ Laravel Mix အတွက်

✓ Local Development Environments (Optional)

- XAMPP / WAMP / MAMP (Traditional)
- Laravel Sail (Docker) (Official Docker Setup)
- Laragon (Lightweight & Fast)
- DevKinsta (For WordPress + Laravel)

✍ Short Notes:

Laravel Development အတွက် အနည်းဆုံး 4GB RAM, PHP 8.1+, Composer, Node.js နှင့် Database တို့ လိုအပ်ပါတယ်။ VS Code / PHPStorm နဲ့ Local Server (Docker, XAMPP) တို့ကို အသုံးပြုနိုင်ပါတယ်။ ဒီ Requirements တွေ ပြည့်မီရင် Laravel Project ကို စတင် Develop လုပ်နိုင်မယ် ဖြစ်ပါတယ်။ ဒီ စာအုပ်မှာ xampp server ကို အသုံးပြုပြီး လေ့ကျင့်ခန်းတွေကို တစ်ခုပြီးတစ်ခု ပြုလုပ်သွားမှာ ဖြစ်ပါတယ်။

Project Installation

Project Installation

Laravel Project တစ်ခုကို အဆင့်ဆင့် Install လုပ်နည်း ကို အသေးစိတ် ရှင်းပြပေးမယ်။

---

1. လိုအပ်သော Software Tools များ

Laravel Install မလုပ်ခင် အောက်ပါ Tools တွေ ပြင်ဆင်ထားပါ။

- PHP 8.1+ (Laravel 10 အတွက်)
- Composer (PHP Dependency Manager)
- Node.js (v16+) (Frontend အတွက်)
- Database (MySQL, SQLite, PostgreSQL)

---

2. Laravel Project အသစ် Install လုပ်နည်း

✓ Composer ဖြင့် Install လုပ်ခြင်း (အလွယ်ဆုံးနည်း)

a) Terminal/CMD ဖွင့်ပါ။
b) အောက်ပါ Command ရိုက်ထည့်ပါ။

Command:

```
composer create-project laravel/laravel project-name
```

Example:

Command:

```
composer create-project laravel/laravel myapp
```

c) Installation ပြီးသည်အထိ စောင့်ပါ (အင်တာနက် အမြန်နှုန်းပေါ်မူတည်ပါတယ်)။

☑ Laravel Installer သုံးပြီး Install လုပ်ခြင်း (Alternative)

a) Laravel Installer ကို Global Install လုပ်ပါ။

Command:

```
composer global require laravel/installer
```

b) Create New Project

Command:

```
laravel new project-name
```

Command:

```
laravel new myshop
```

☑ Project Setup အပြည့်အစုံ

✓ Environment File (.env) ပြင်ဆင်ခြင်း

1. Project Folder ထဲဝင်ပါ။

Command:

```
cd myapp
```

2.env.example ကို Copy ကူးပြီး .env အဖြစ် ပြောင်းပါ။

Command:

```
cp .env.example .env
```

✓ Application Key Generate လုပ်ပါ။

Command:

```
php artisan key:generate
```

✓ Database Configuration

1. `.env` ဖိုင်ကို ဖွင့်ပြီး Database Connection ပြင်ပါ။

Code:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel
DB_USERNAME=root
DB_PASSWORD=
```

2. Database Create လုပ်ပါ (MySQL ဆိုရင် phpMyAdmin သုံးနိုင်ပါတယ်)။

✓ Frontend Dependencies Install လုပ်ခြင်း

1. **Node Modules Install**

Command:

```
npm install
```

2. **Development Mode ဖြင့် Run**

Command:

```
npm run dev
```

☑ Development Server Start လုပ်ခြင်း

1. Terminal မှာ အောက်ပါ Command ရိုက်ပါ။

   Command:

```
php artisan serve
```

2. Browser ဖွင့်ပြီး အောက်ပါ Link ကို ရိုက်ထည့်ပါ။

   Code:

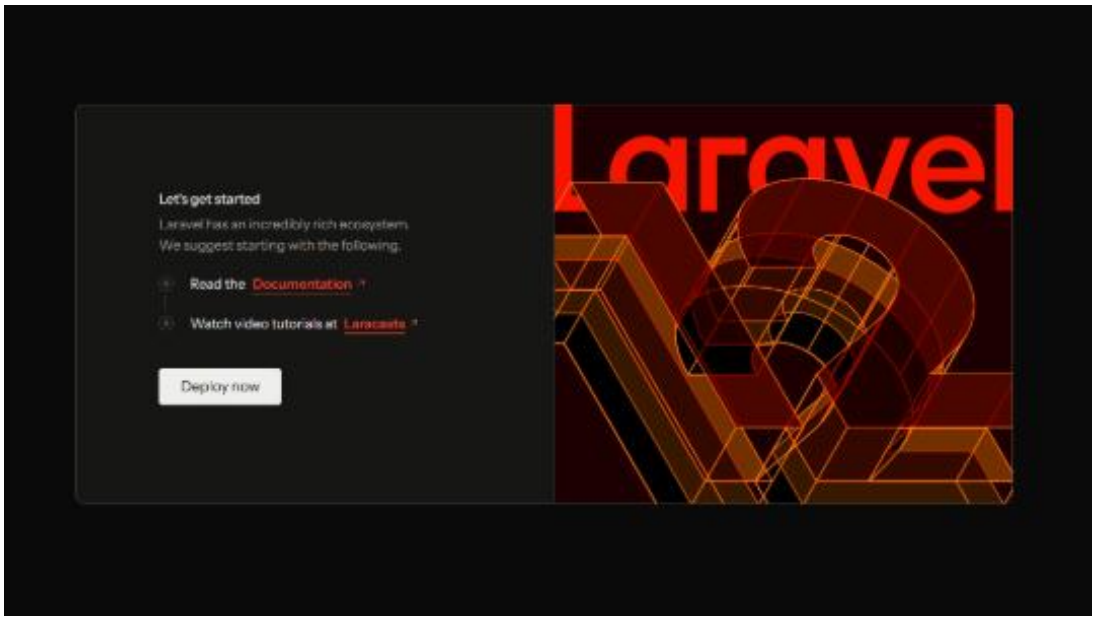
```
http://localhost:8000
```

   - Laravel Default Page ပေါ်ရင် Installation Successful!

Laravel Project Install လုပ်ဖို့ **Composer, PHP 8.1+, Node.js** တို့ လိုအပ်ပါတယ်။

- composer create-project laravel/laravel project-name → Project Create
- php artisan serve → Local Server Start

http://localhost:3000



Project Folers

Project Folders

Laravel project တစ်ခုကို create လုပ်လိုက်တဲ့အခါမှာ အောက်ပါ folder တွေ အလိုအလျောက် ဖန်တီးပေးပါတယ်။ အဓိက folder တစ်ခုချင်းစီရဲ့ အလုပ်လုပ်ပုံကို ရှင်းပြပေးမယ်။ Laravel project folder structure ကို ကောင်းစွာ နားလည်ထားပါက project ကို ပိုမိုထိရောက်စွာ manage လုပ်နိုင်မယ် ဖြစ်ပါတယ်။

✓ အဓိက Folder များ

app/ - Application Core Logic

- Controllers/ - Request များကို handle လုပ်ပြီး Model နဲ့ View ကို ချိတ်ဆက်ပေးတဲ့ file များ ဖြစ်ပါတယ်။
- Models/ - Database tables များနဲ့ ချိတ်ဆက်အလုပ်လုပ်မယ့် files တွေဖြစ်ပါတယ်။
- Providers/ - Service registration နဲ့ dependency injection တွေအတွက်ဖြစ်ပါတယ်။
- Http/ - Middleware, Requests နဲ့ Responses များအတွက်ဖြစ်ပါတယ်။

bootstrap/ - Framework Startup Files

- app.php - Laravel application စတင်တဲ့အခါ load လုပ်မယ့် file ဖြစ်ပါတယ်။

config/ - Configuration Files

- app.php - Application settings
- database.php - Database connection settings
- mail.php - Email configuration

database/ - Database Related Files

- migrations/ - Database tables များ create/update လုပ်ဖို့ ဖြစ်ပါတယ်။
- seeders/ - Dummy data များ insert လုပ်ဖို့ ဖြစ်ပါတယ်။
- factories/ - Test data များ generate လုပ်ဖို့ ဖြစ်ပါတယ်။

public/ - Publicly Accessible Files

- index.php - Application entry point အတွက် ဖြစ်ပါတယ်။
- CSS, JS နဲ့ image files များ သိမ်းဆည်းရန် ဖြစ်ပါတယ်။

resources/ - Application Assets

- views/ - Blade template files (HTML)
- lang/ - Language files
- sass/ - CSS preprocessor files

routes/ - Application Routes

- web.php - Web routes
- api.php - API routes
- console.php - Artisan command routes

storage/ - Storage Files

- app/ - Application generated files
- framework/ - Framework generated files
- logs/ - Error log files

tests/ - Test Files

- Feature tests နဲ့ Unit tests များအတွက်

vendor/ - Composer Dependencies

- Composer ဖြင့် install လုပ်ထားသော packages များ

✓ အခြား အရေးကြီးသော Files

.env - Environment Configuration

- Database credentials, app key နဲ့ အခြား sensitive settings များ

artisan - Command Line Tool

- Laravel commands များ run ဖို့

composer.json - PHP Dependencies

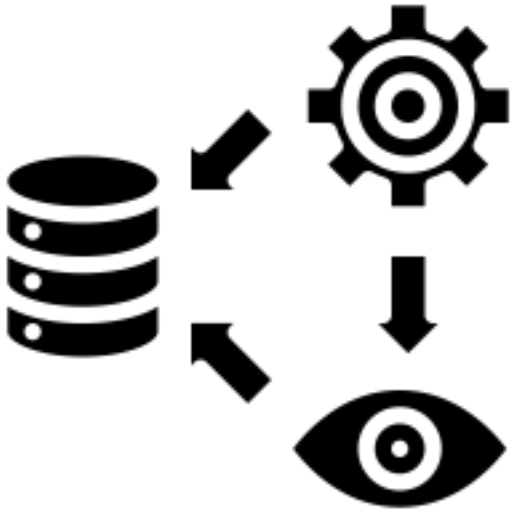
- Project အတွက် required packages များ

✓ Folder Structure ကို ဘာကြောင့် သိထားသင့်သလဲ?

1. Code Organization - File တွေကို စနစ်တကျ ခွဲခြားသိမ်းဆည်းနိုင်ခြင်း
2. Team Collaboration - Developer အသစ်များ project ကို လွယ်ကူစွာ နားလည်နိုင်ခြင်း
3. Maintenance - File များကို အလွယ်တကူ ရှာဖွေပြင်ဆင်နိုင်ခြင်း

စတဲ့ အကျိုးကျေးဇူးတွေကို ရရှိမှာ ဖြစ်ပါတယ်။

MVC



MVC

MVC (Model-View-Controller) Pattern ကို အခြေခံထားသော PHP Framework တစ်ခု ဖြစ်ပါတယ်။ MVC ကို အသုံးပြုခြင်းဖြင့် Code များကို စနစ်တကျ ခွဲခြားရေးသားနိုင်ပြီး၊ Web Application Development ကို ပိုမိုလွယ်ကူမြန်ဆန်စေပါတယ်။

✓ MVC ဆိုတာ ဘာလဲ?

MVC ကို Model, View, Controller (၃) ပိုင်း ခွဲထားပါတယ်။

အပိုင်း	အလုပ်လုပ်ပုံ
Model	Database နှင့် တိုက်ရိုက်ချိတ်ဆက်ပြီး Data များကို ကိုင်တွယ်ပါတယ်။
View	User မြင်ရမယ့် UI (HTML, CSS, Blade Template) ကို ဖော်ပြပါတယ်။
Controller	User Request များကို လက်ခံပြီး Model & View များနှင့် ချိတ်ဆက်ပေးပါတယ်။

✓ Laravel MVC ဘယ်လို အလုပ်လုပ်သလဲ?

အဆင့် (၁) - User က Request လုပ်ခြင်း

- User က Browser မှာ URL (e.g., /products) ကို ရိုက်ထည့်ပါတယ်။

အဆင့် (၂) - Route မှတဆင့် Controller သို့ လမ်းညွှန်ခြင်း

- routes/web.php ဖိုင်မှာ Request ကို Controller ဆီ ပို့ပါတယ်။

Code:

```
Route::get('/products', [ProductController::class, 'index']);
```

အဆင့် (၃) - Controller က Model နှင့် တွဲအလုပ်လုပ်ခြင်း

- ProductController က Product Model မှ Data များကို ယူပါတယ်။

Code:

```
public function index() {
    $products = Product::all(); // Model မှ Data များကို ယူခြင်း
    return view('products.index', ['products' => $products]); // View သို့ Dat
a များ ပို့ခြင်း
}
```

အဆင့် (၄) - View က Data များကို ဖော်ပြခြင်း

- resources/views/products/index.blade.php ဖိုင်မှာ Data များကို ဖော်ပြပါတယ်။

Code:

```
@foreach ($products as $product)
    <h3>{{ $product->name }}</h3>
@endforeach
```

အဆင့် (၅) - User ထံ HTML Response ပြန်ပို့ခြင်း

- Laravel က View မှ ပြင်ဆင်ထားသော HTML ကို User ထံ ပြန်ပို့ပါတယ်။

✓ Laravel MVC ၏ အားသာချက်များ

- ✅ Code Organization → Model, View, Controller ကို သီးသန့်ခွဲထားသဖြင့် ပြင်ဆင်ရလွယ်ပါတယ်။
- ✅ Reusability → Model နှင့် Controller များကို နေရာစုံမှာ ပြန်သုံးနိုင်ပါတယ်။
- ✅ Security → Database Logic ကို Model တွင်သာ ရေးသဖြင့် SQL Injection အန္တရာယ် လျော့နည်းပါတယ်။
- ✅ Team Collaboration → Developer, Designer တို့အတွက် သီးသန့် အပိုင်းများ ခွဲထားနိုင်ပါတယ်။

☑ Laravel MVC ကို ဘယ်လို စတင်သုံးမလဲ?

✓ Model Create လုပ်ခြင်း

Command:

```
php artisan make:model Product -m
```

- `-m` flag က Migration File (Database Table) ကိုပါ တစ်ခါတည်း Create လုပ်ပေးပါတယ်။

✓ Controller Create လုပ်ခြင်း

Command:

```
php artisan make:controller ProductController
```

✓ View Create လုပ်ခြင်း

- `resources/views/products/index.blade.php` ဖိုင်ကို ဖန်တီးပြီး HTML/Blade Template ရေးပါ။

✓ Laravel MVC Flow အလုပ်လုပ်ပုံ အသေးစိတ်

1. User → /products ကို Request လုပ်မယ်။
2. Route → ProductController@index ကို လှမ်းခေါ်မယ်။
3. Controller → Product::all() ဖြင့် Database မှ Data များယူမယ်။
4. View → products.index.blade.php မှာ Data များဖော်ပြမယ်။
5. User → Product List ကို Browser မှာ မြင်ရမယ်။

Laravel MVC Architecture သည် Web App Development ကို လွယ်ကူမြန်ဆန်စေပြီး Code များကို စနစ်တကျ စီမံနိုင်စေပါတယ်။

- Model → Database နှင့် အလုပ်လုပ်ခြင်း
- View → UI ဖော်ပြခြင်း
- Controller → Logic များကို ထိန်းချုပ်ခြင်း

MVC ကို ကောင်းစွာ နားလည်ထားပါက Laravel Project များကို ပိုမိုထိရောက်စွာ တည်ဆောက်နိုင်မယ် ဖြစ်ပါတယ်။ ဆက်လက်ပြီး Model ၊ View ၊ Controller အကြောင်းတွေကို အသေးစိတ် လေ့လာကြည့်ကြရအောင်။

☑ Model

Laravel မှာ **Model** ဆိုတာ database tables တွေနဲ့ interact လုပ်ဖို့အတွက် အဓိက class တစ်ခုဖြစ်ပါတယ်။ Eloquent ORM ကို အသုံးပြုပြီး database operations တွေကို လွယ်ကူစွာ လုပ်ဆောင်နိုင်ပါတယ်။ Model Relationships တွေကို တော့ နောက်လာမယ့်လေ့ကျင့်ခန်း တွေမှာ ဆက်ပြီး လေ့လာရမှာ ဖြစ်ပါတယ်။

✓ Model ဆိုတာ ဘာလဲ?

Model သည်:

- Database table တစ်ခုနဲ့ တိုက်ရိုက်ချိတ်ဆက်ထားသော class
- Eloquent ORM ကို အသုံးပြုထားခြင်း
- Business logic များကို သိမ်းဆည်းရန်

✓ Model တစ်ခု ဘယ်လိုဖန်တီးမလဲ

Command:

```
php artisan make:model Product
```

- app/Models/Product.php ဖိုင်ကို ဖန်တီးပေးမယ်
- Migration ပါတစ်ခါတည်း ဖန်တီးချင်ရင်:

Command:

```
php artisan make:model Product -m
```

✓ Model ၏ အခြေခံ Structure

Code:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Product extends Model
{
    // Model အတွက် configurations
}
```

✓ Model Configuration တွေဘယ်လို လုပ်မလဲ

☐ Table Name သတ်မှတ်ခြင်း

Code:

```
protected $table = 'my_products';
```

- Default: Model name ကို plural လုပ်ပြီး ယူမယ် (Product → products)

☐ Primary Key သတ်မှတ်ခြင်း

Code:

```
protected $primaryKey = 'product_id';
```

- Default: `id` column

☐ Timestamps ဖြုတ်ခြင်း

Code:

```
public $timestamps = false;
```

- Default: created_at နဲ့ updated_at columns ကို အလိုအလျောက် update လုပ်မယ်

☐ Fillable Fields သတ်မှတ်ခြင်း

Code:

```
protected $fillable = ['name', 'price', 'description'];
```

- Mass assignment လုပ်လို့ရတဲ့ columns တွေ

☐ Guarded Fields သတ်မှတ်ခြင်း

Code:

```
protected $guarded = ['id'];
```

- Mass assignment လုပ်လို့မရတဲ့ columns တွေ

✓ Model အသုံးပြုပြီး Database Operations များ ဘယ်လို ပြုလုပ်မလဲ

☐ Data အားလုံးရယူခြင်း

Code:

```
$products = Product::all();
```

☐ Condition နဲ့ရယူခြင်း

Code:

```
$expensiveProducts = Product::where('price', '>', 100)->get();
```

☐ Single Record ရယူခြင်း

Code:

```
$product = Product::find(1);
```

☐ New Record ထည့်ခြင်း

Code:

```
Product::create([
    'name' => 'New Product',
    'price' => 99.99
]);
```

☐ Record Update လုပ်ခြင်း

Code:

```
$product = Product::find(1);
$product->price = 150;
$product->save();
```

☐ Record ဖျက်ခြင်း

Code:

```
Product::destroy(1);
// or
$product = Product::find(1);
$product->delete();
```

Laravel Model သည် database interaction များကို အလွယ်တကူ လုပ်ဆောင်နိုင်စေပါတယ်။ Eloquent ORM ၏ အားသာချက်များကို အသုံးပြုပြီး clean ဖြစ်တဲ့ code များ ရေးသားနိုင်ပါတယ်။

☑ View

Laravel မှာ View ဆိုတာ application ၏ user interface (UI) ကို ဖော်ပြတဲ့ အစိတ်အပိုင်း ဖြစ်ပါတယ်။ Views များကို resources/views folder အတွင်းမှာ သိမ်းဆည်း ထားပြီး .blade.php extension နဲ့ ရေးသားလေ့ရှိပါတယ်။

✓ View ဆိုတာ ဘာလဲ?

View သည် HTML code တွေ နဲ့ User Interface (UI) ကို ဖော်ပြရန် သတ်မှတ်ထား တဲ့နေရာဖြစ်ပါတယ်။ Controller မှ ပို့လိုက်တဲ့ data များကို display လုပ်ပေးရပါတယ်။ ဒီလို data တွေကို display လုပ်တဲ့အခါ Blade Template Engine ကို အသုံးပြုထားပါတယ်။ အဲဒါကြောင့် filename.blade.php extension နဲ့ သိမ်းပေးရပါမယ်။

✓ View တစ်ခု ဖန်တီးနည်း

Code:

```
resources/views/welcome.blade.php
```

- File name: welcome.blade.php (Blade template ဖြစ်ကြောင်း ဖော်ပြခြင်း)

✓ View ကို Controller မှ ခေါ်ယူနည်း

Code:

```
// Controller ထဲမှာ
public function showWelcome()
{
    return view('welcome');
}
```

- view() helper function ကို အသုံးပြုပြီး view ကို return ပြန်မယ်

✓ View သို့ Data ပို့တဲ့အခါ ပုံစံအမျိုးမျိုးနဲ့ ပို့လို့ရပါတယ်။

☐ Array အဖြစ် ပို့နည်း

Code:

```
return view('welcome', ['name' => 'John']);
```

☐ with() method အသုံးပြုနည်း

Code:

```
return view('welcome')->with('name', 'John');
```

☐ Compact အသုံးပြုနည်း

Code:

```
$name = 'John';
$age = 25;
return view('welcome', compact('name', 'age'));
```

✓ View Composers အသုံးပြုနည်း

Code:

```
// AppServiceProvider မှာ
View::composer('profile', function ($view) {
    $view->with('count', User::count());
});
```

- profile view ကို ခေါ်တိုင်း count variable ကို အလိုအလျောက် ထည့်ပေးမယ်

✓ View များကို စစ်ဆေးနည်း

Command:

```
php artisan view:clear
```

- Compiled views များကို clear လုပ်မယ်

✓ View များကို Cache လုပ်ခြင်း

Command:

```
php artisan view:cache
```

- Views များကို cache လုပ်ပြီး performance ကောင်းစေမယ်

☑ Controller

Laravel မှာ **Controller** ဆိုတာ application logic များကို စီမံခန့်ခွဲတဲ့ အဓိက class တစ်ခုဖြစ်ပါတယ်။ Controller များကို app/Http/Controllers folder အတွင်းမှာ သိမ်းဆည်းထားပါတယ်။

✓ Controller ဆိုတာ ဘာလဲ?

Controller သည် User requests များကို လက်ခံပြီး process လုပ်ပေးမယ်။ Model နဲ့ View ကို ချိတ်ဆက်ပေးမယ်။ ပြီးတော့ application project တစ်ခုလုံးရဲ့ Business logic များကို စီမံခန့်ခွဲပေးတဲ့ class file တွေဖြစ်ပါတယ်။

✓ Controller တစ်ခု ဖန်တီးနည်း

Command:

```
php artisan make:controller ProductController
```

- app/Http/Controllers/ProductController.php ဖိုင်ကို ဖန်တီးပေးမယ်

Resource Controller ဖန်တီးချင်ရင်:

Command:

```
php artisan make:controller ProductController --resource
```

- CRUD operations အတွက် method များ အလိုအလျောက် ပါဝင်မယ်

✓ Controller ၏ အခြေခံ Structure

Code:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class ProductController extends Controller
{
    // Controller methods များ
}
```

✓ Resource Controller Methods

Method	URI	Action	Route Name
index	GET /products	စာရင်းအားလုံးပြမယ်	products.index
create	GET /products/create	အသစ်ဖန်တီးဖို့ form ပြမယ်	products.create
store	POST /products	အသစ်သိမ်းမယ်	products.store
show	GET /products/{id}	တစ်ခုတည်းပြမယ်	products.show
edit	GET /products/{id}/edit	တည်းဖြတ်ဖို့ form ပြမယ်	products.edit
update	PUT/PATCH /products/{id}	update လုပ်မယ်	products.update
destroy	DELETE /products/{id}	ဖျက်မယ်	products.destroy

✓ Controller မှ Route သို့ Data ပို့နည်း

☐ View သို့ Data ပို့ခြင်း

Code:

```
public function show($id)
{
    $product = Product::find($id);
    return view('products.show', compact('product'));
}
```

☐ JSON Response ပြန်ခြင်း

Code:

```
public function apiIndex()
{
    return response()->json(Product::all());
}
```

☐ Redirect လုပ်ခြင်း

Code:

```
public function store(Request $request)
{
    // Save logic
    return redirect()->route('products.index');
}
```

✓ Dependency Injection အသုံးပြုနည်း

Code:

```
use App\Models\Product;

public function show(Product $product)
{
    return view('products.show', compact('product'));
}
```

- Route Model Binding အလိုအလျောက် လုပ်ဆောင်ပေးမယ်

✓ Request Handling

Code:

```
use Illuminate\Http\Request;

public function store(Request $request)
{
    $validated = $request->validate([
        'name' => 'required|max:255',
        'price' => 'required|numeric'
    ]);

    Product::create($validated);

    return redirect('/products');
}
```

✓ Controller Middleware အသုံးပြုနည်း

☐ Constructor မှာ သတ်မှတ်နည်း

Code:

```
public function __construct()
{
    $this->middleware('auth');
    $this->middleware('log')->only('index');
    $this->middleware('subscribed')->except('store');
}
```

☐ Route မှာ တိုက်ရိုက်သတ်မှတ်နည်း

Code:

```
Route::get('/profile', [UserController::class, 'show'])
    ->middleware('auth');
```

✓ Create Single Action Controller

Command:

```
php artisan make:controller ShowProfile --invokable
```

Code:

```
public function __invoke()
{
    return view('profile');
}
```

Code:

```
Route::get('/profile', ShowProfile::class);
```

✓ Controller Best Practices

1. Thin Controllers, Fat Models - Business logic များကို Model မှာ ရေးပါ
2. Resource Controllers ကို အသုံးပြုပါ
3. Form Requests ကို အသုံးပြုပြီး validation လုပ်ပါ
4. Type Hinting ကို အသုံးပြုပါ
5. Method Injection ကို အသုံးပြုပါ

Laravel Controllers များသည် application ၏ logic flow ကို စနစ်တကျ စီမံခန့်ခွဲနိုင်ပါတယ်။ Controller များကို ကောင်းစွာ နားလည်ထားပါက Laravel application များကို ပိုမိုကောင်းမွန်စွာ တည်ဆောက်နိုင်မယ် ဖြစ်ပါတယ်။ Model View Controller တွေကို နားလည် သဘောပေါက်သွားပြီး ဆိုရင် တော့ blade template engine အကြောင်းကို ဆက်ပြီး လေ့လာရအောင်။

BLADE TEMPLATE ENGINE

Blade Template Engine

Laravel Blade သည် Laravel framework မှာပါဝင်တဲ့ powerful template engine တစ်ခုဖြစ်ပါတယ်။ Blade ကို အသုံးပြုခြင်းဖြင့် PHP code တွေကို clean ဖြစ်ပြီး readable ဖြစ်အောင် ရေးနိုင်မယ်။ Blade file တွေရဲ့ extension က .blade.php ဖြစ်ပြီး views ဖိုလ်ဒါထဲမှာ သိမ်းဆည်းရပါမယ်။

☑ Blade ၏ အဓိကလုပ်ဆောင်ချက်များ

✓ Template Inheritance

Blade မှာ layout system တစ်ခုပါဝင်ပြီး template inheritance ကို အသုံးပြုနိုင်ပါတယ်။

⇨ master layout ဖန်တီးခြင်း

Code:

```
<!-- resources/views/layouts/master.blade.php -->
<html>
    <head>
        <title>@yield('title')</title>
    </head>
    <body>
        @section('sidebar')
            ဒါက default sidebar content ဖြစ်ပါတယ်
        @show

        <div class="container">
            @yield('content')
        </div>
    </body>
</html>
```

⇨ child view ဖန်တီးခြင်း

Code:

```
<!-- resources/views/child.blade.php -->
@extends('layouts.master')

@section('title', 'Page Title')

@section('sidebar')
    @parent
    <p>ဒါက sidebar ကို extend လုပ်ထားတာဖြစ်ပါတယ်</p>
@endsection

@section('content')
    <p>ဒါက main content ဖြစ်ပါတယ်</p>
@endsection
```

✓ Data Displaying

Blade မှာ data တွေကို display လုပ်ဖို့ `{{ }}` syntax ကို အသုံးပြုပါတယ်။

Code:

```
Hello, {{ $name }}!

Current UNIX timestamp: {{ time() }}
```

✓ Control Structures

Blade မှာ if statements, loops တို့အတွက် clean syntax တွေပါဝင်ပါတယ်။

⇨ If Statements

Code:

```
@if (count($records) === 1)
    ဒါက record တစ်ခုတည်းပါ
@elseif (count($records) > 1)
    ဒါက records အများကြီးပါ
@else
    ဒါက record မရှိပါ
@endif
```

⇨ Loops

Code:

```
@for ($i = 0; $i < 10; $i++)
    Current value is {{ $i }}
@endfor

@foreach ($users as $user)
    <p>ဒါက user {{ $user->id }} ဖြစ်ပါတယ်</p>
@endforeach

@while (true)
    <p>ဒါက infinite loop ဖြစ်ပါတယ်</p>
@endwhile
```

✓ Components & Slots

Blade မှာ components တွေကို အသုံးပြုနိုင်ပါတယ်။

⇨ Component ဖန်တီးခြင်း

Code:

```
<!-- resources/views/components/alert.blade.php -->
<div class="alert alert-danger">
    {{ $slot }}
</div>
```

⇨ Component ကို အသုံးပြုခြင်း

Code:

```
@component('components.alert')
    <strong>သတိထားပါ!</strong> ဒါက error message တစ်ခုဖြစ်ပါတယ်
@endcomponent
```

✓ Including Sub-Views

Blade မှာ တခြား view တွေကို include လုပ်နိုင်ပါတယ်။

Code:

```
@include('shared.errors')

@include('view.name', ['some' => 'data'])
```

✓ Service Injection

Blade မှာ service container ကနေ service တွေကို inject လုပ်နိုင်ပါတယ်။

Code:

```
@inject('metrics', 'App\Services\MetricsService')

<div>
    Monthly Revenue: {{ $metrics->monthlyRevenue() }}
</div>
```

✓ Blade ရဲ့အခြားအသုံးဝင်တဲ့ Features များ

⇨ Raw PHP

Blade ထဲမှာ raw PHP code တွေရေးဖို့ `@php` directive ကို အသုံးပြုနိုင်ပါတယ်။

Code:

```
@php
    $counter = 1;
@endphp
```

⇨ Comments

Blade comments က HTML comments နဲ့မတူပါဘူး။ Blade comments က rendered HTML ထဲမှာ မပါဝင်ပါဘူး။

Code:

```
{{-- ဒါက Blade comment ဖြစ်ပါတယ် --}}
```

3. Custom Directives

Blade မှာ custom directives တွေကို ဖန်တီးနိုင်ပါတယ်။

Service Provider ထဲမှာ

Code:

```
Blade::directive('datetime', function ($expression) {
    return "<?php echo ($expression)->format('m/d/Y H:i'); ?>";
});
```

Blade File ထဲမှာ အသုံးပြုခြင်း

Code:

```
@datetime($date)
```

Laravel Blade Template Engine သည် clean ဖြစ်ပြီး powerful ဖြစ်တဲ့ template system တစ်ခုဖြစ်ပါတယ်။ Template inheritance, control structures, components တို့ကို အသုံးပြုခြင်းဖြင့် application တွေရဲ့ UI ကို ပိုမိုလွယ်ကူစွာ ဖန်တီးနိုင်ပါတယ်။ Blade သည် PHP code တွေကို cache လုပ်ထားတာကြောင့် performance လည်းကောင်းမွန်ပါတယ်။

ROUTE

Routes

Laravel မှာ Route ဆိုတာ URL တစ်ခုနဲ့ Application Logic တစ်ခုကို ချိတ်ဆက်ပေးတဲ့ နည်းလမ်းဖြစ်ပါတယ်။ အောက်မှာ Laravel Route များအကြောင်း အသေးစိတ် ရှင်းပြပေးမယ်။

✓ Basic Route ရေးနည်း

❖ Simple Route

Code:

```
Route::get('/hello', function () {
    return 'မင်္ဂလာပါ Laravel!';
});
```

- အလုပ်လုပ်ပုံ: /hello URL ကို ဝင်ရောက်လာရင် "မင်္ဂလာပါ Laravel!" ဆိုတဲ့ message ပြန်ပို့မယ်

❖ View Return ပြန်ပို့ခြင်း

Code:

```
Route::get('/welcome', function () {
    return view('welcome');
});
```

- အလုပ်လုပ်ပုံ: /welcome URL ကို ဝင်ရောက်လာရင် resources/views/welcome.blade.php ဖိုင်ကို ပြမယ်

✓ Route Parameters အသုံးပြုနည်း

❖ Basic Parameter

Code:

```
Route::get('/user/{id}', function ($id) {
```

```
    return 'User ID: '.$id;
});
```

- အလုပ်လုပ်ပုံ: /user/5 ဆိုရင် "User ID: 5" ပြန်ပို့မယ်

❖ Optional Parameter

Code:

```
Route::get('/user/{name?}', function ($name = 'Guest') {
    return 'User Name: '.$name;
});
```

- အလုပ်လုပ်ပုံ: /user ဆိုရင် "User Name: Guest" ပြန်ပို့မယ်

✓ Route များကို Controller နဲ့ ချိတ်ဆက်နည်း

Code:

```
Route::get('/products', 'ProductController@index');
```

- အလုပ်လုပ်ပုံ: /products URL ကို
  ဝင်ရောက်လာရင် ProductController ထဲက index method ကို ခေါ်မယ်

✓ Route Groups အသုံးပြုနည်း

❖ Prefix အသုံးပြုခြင်း

Code:

```
Route::prefix('admin')->group(function () {
    Route::get('/users', function () {
        return 'Admin Users Page';
    });
});
```

- အလုပ်လုပ်ပုံ: /admin/users URL ကို ဝင်ရောက်လာရင် "Admin Users Page" ပြန်ပို့မယ်

❖ Middleware အသုံးပြုခြင်း

Code:

```
Route::middleware(['auth'])->group(function () {
    Route::get('/profile', function () {
        return 'User Profile';
    });
});
```

- အလုပ်လုပ်ပုံ: /profile URL ကို ဝင်ရောက်လာရင် auth middleware ကို အရင် စစ်ဆေးမယ်

✓ Route Naming အသုံးပြုနည်း

Code:

```
Route::get('/user/profile', function () {
    //
})->name('profile');
```

- အသုံးပြုနည်း: route('profile') ဆိုပြီး URL generate လုပ်နိုင်မယ်

Code:

```
Route::get('/profile','UserController::class','profile')->name('profile');
```

- အသုံးပြုနည်း: route('profile') ဆိုပြီး URL generate လုပ်နိုင်မယ်

✓ Resource Routes အသုံးပြုနည်း

Code:

```
Route::resource('products', 'ProductController');
```

- **အလုပ်လုပ်ပုံ:** CRUD operations အတွက် အလိုအလျောက် routes 7 ခု ဖန်တီးပေးမယ်

HTTP Method	URI	Action	Route Name
GET	/products	index	products.index
GET	/products/create	create	products.create
POST	/products	store	products.store
GET	/products/{product}	show	products.show
GET	/products/{product}/edit	edit	products.edit
PUT/PATCH	/products/{product}	update	products.update
DELETE	/products/{product}	destroy	products.destroy

✓ Route Caching အကြောင်း

Code:

```
php artisan route:cache
```

- အသုံးဝင်ပုံ: Production environment မှာ Route performance ကို မြန်ဆန်စေမယ်

✓ Route များကို ဘယ်မှာ ရေးမလဲ?

- Web Routes: `routes/web.php`
- API Routes: `routes/api.php`

✓ အသုံးများသော Route Methods

Method	အသုံးပြုပုံ
`Route::get()`	GET request များအတွက်
`Route::post()`	POST request များအတွက်
`Route::put()`	PUT request များအတွက်
`Route::patch()`	PATCH request များအတွက်
`Route::delete()`	DELETE request များအတွက်
`Route::any()`	မည်သည့် HTTP method မဆို

✓ Route များကို စမ်းသပ်နည်း

Command:

```
php artisan route:list
```

- **အလုပ်လုပ်ပုံ:** Application ထဲမှာ ရှိတဲ့ Route အားလုံးကို list ထုတ်ပြမယ်

Laravel Routing သည် Web Application တစ်ခုအတွက် အလွန်အရေးကြီးပါတယ်။ Route များကို ကောင်းစွာ နားလည်ထားပါက Application ကို ပိုမိုကောင်းမွန်စွာ တည်ဆောက်နိုင်မယ် ဖြစ်ပါတယ်။

CRUD

CRUD

CRUD Features များ

1. Create (ဖန်တီးခြင်း) - ကုန်ပစ္စည်းအသစ်ထည့်ခြင်း
2. Read (ဖတ်ရှုခြင်း) - ကုန်ပစ္စည်းစာရင်းကြည့်ခြင်း
3. Update (ပြင်ဆင်ခြင်း) - ကုန်ပစ္စည်းအချက်အလက်ပြင်ခြင်း
4. Delete (ဖျက်ခြင်း) - ကုန်ပစ္စည်းဖျက်ခြင်း

1. Project စတင်ခြင်း

ပထမဆုံး Laravel project အသစ်တစ်ခုစတင်ပါမယ်။

Command:

```
composer create-project laravel/laravel product-crud
cd product-crud
```

2. Database ချိတ်ဆက်ခြင်း

.env ဖိုင်ကိုပြင်ဆင်ပါမယ်။

Code:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=product_crud
DB_USERNAME=root
DB_PASSWORD=
```

3. Product Model နှင့် Migration ဖန်တီးခြင်း

Code:

```
php artisan make:model Product -m
```

database/migrations/xxxx_create_products_table.php ဖိုင်ကိုပြင်ဆင်ပါမယ်။

Code:

```
public function up()
{
    Schema::create('products', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->decimal('price', 8, 2);
        $table->timestamps();
    });
}
```

Migration ကို run ပါမယ်။

Code:

```
php artisan migrate
```

4. Product Model ပြင်ဆင်ခြင်း

app/Models/Product.php ဖိုင်ကိုပြင်ဆင်ပါမယ်။

Code:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Product extends Model
{
    use HasFactory;

    protected $fillable = [
        'name',
        'price'
    ];
}
```

5. Product Controller ဖန်တီးခြင်း

Command:

```
php artisan make:controller ProductController --resource
```

app/Http/Controllers/ProductController.php ဖိုင်ကိုပြင်ဆင်ပါမယ်။

Code:

```
<?php

namespace App\Http\Controllers;

use App\Models\Product;
use Illuminate\Http\Request;

class ProductController extends Controller
{
    // Product List
    public function index()
    {
        $products = Product::all();
        return view('products.index', compact('products'));
    }

    // Create Form
    public function create()
    {
        return view('products.create');
    }

    // Store Product
    public function store(Request $request)
    {
        $request->validate([
            'name' => 'required',
            'price' => 'required|numeric'
        ]);

        Product::create($request->all());

        return redirect()->route('products.index')
                         ->with('success', 'Product created successfully.')
;
    }

    // Show Product
    public function show(Product $product)
    {
        return view('products.show', compact('product'));
    }
```

```
    // Edit Form
    public function edit(Product $product)
    {
        return view('products.edit', compact('product'));
    }

    // Update Product
    public function update(Request $request, Product $product)
    {
        $request->validate([
            'name' => 'required',
            'price' => 'required|numeric'
        ]);

        $product->update($request->all());

        return redirect()->route('products.index')
                         ->with('success', 'Product updated successfully.')
;
    }

    // Delete Product
    public function destroy(Product $product)
    {
        $product->delete();

        return redirect()->route('products.index')
                         ->with('success', 'Product deleted successfully.')
;
    }
}
```

6. Views ဖိုင်များဖန်တီးခြင်း

Layout File (resources/views/layouts/app.blade.php)

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>Product CRUD</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/boots
trap.min.css" rel="stylesheet">
</head>
<body>
    <div class="container mt-4">
        @yield('content')
    </div>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/boots
trap.bundle.min.js"></script>
</body>
</html>
```

Index Page (resources/views/products/index.blade.php)

Code:

```
@extends('layouts.app')

@section('content')
    <div class="d-flex justify-content-between align-items-center mb-4">
        <h1>Product List</h1>
        <a href="{{ route('products.create') }}" class="btn btn-primary">Ad
d Product</a>
    </div>

    @if ($message = Session::get('success'))
        <div class="alert alert-success">
            {{ $message }}
        </div>
    @endif

    <table class="table table-bordered">
        <thead>
            <tr>
                <th>ID</th>
                <th>Name</th>
                <th>Price</th>
                <th width="280px">Action</th>
            </tr>
        </thead>
        <tbody>
            @foreach ($products as $product)
            <tr>
                <td>{{ $product->id }}</td>
```

```
                <td>{{ $product->name }}</td>
                <td>${{ number_format($product->price, 2) }}</td>
                <td>
                    <form action="{{ route('products.destroy',$product->id)
}}" method="POST">
                        <a class="btn btn-info" href="{{ route('products.sh
ow',$product->id) }}">Show</a>
                        <a class="btn btn-primary" href="{{ route('products
.edit',$product->id) }}">Edit</a>
                        @csrf
                        @method('DELETE')
                        <button type="submit" class="btn btn-danger">Delete
</button>
                    </form>
                </td>
            </tr>
            @endforeach
        </tbody>
    </table>

@endsection
```

Create Form (resources/views/products/create.blade.php)

Code:

```
@extends('layouts.app')

@section('content')
    <div class="row">
        <div class="col-lg-12 margin-tb">
            <div class="pull-left">
                <h2>Add New Product</h2>
            </div>
            <div class="pull-right">
                <a class="btn btn-primary" href="{{ route('products.index')
}}"> Back</a>
            </div>
        </div>
    </div>

    @if ($errors->any())
        <div class="alert alert-danger">
            <strong>Whoops!</strong> There were some problems with your inp
ut.<br><br>
            <ul>
```

```
                @foreach ($errors->all() as $error)
                    <li>{{ $error }}</li>
                @endforeach
            </ul>
        </div>
    @endif

    <form action="{{ route('products.store') }}" method="POST">
        @csrf

        <div class="row">
            <div class="col-xs-12 col-sm-12 col-md-12">
                <div class="form-group">
                    <strong>Name:</strong>
                    <input type="text" name="name" class="form-control" pla
ceholder="Name">
                </div>
            </div>
            <div class="col-xs-12 col-sm-12 col-md-12">
                <div class="form-group">
                    <strong>Price:</strong>
                    <input type="number" step="0.01" name="price" class="fo
rm-control" placeholder="Price">
                </div>
            </div>
            <div class="col-xs-12 col-sm-12 col-md-12 text-center">
                <button type="submit" class="btn btn-primary">Submit</butto
n>
            </div>
        </div>
    </form>
@endsection
```

Show Page (resources/views/products/show.blade.php)

Code:

```
@extends('layouts.app')

@section('content')
    <div class="row">
        <div class="col-lg-12 margin-tb">
            <div class="pull-left">
                <h2>Show Product</h2>
            </div>
```

```
            <div class="pull-right">
                <a class="btn btn-primary" href="{{ route('products.index')
}}"> Back</a>
            </div>
        </div>
    </div>

    <div class="row">
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group">
                <strong>Name:</strong>
                {{ $product->name }}
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group">
                <strong>Price:</strong>
                ${{ number_format($product->price, 2) }}
            </div>
        </div>
    </div>
@endsection
```

Edit Form (resources/views/products/edit.blade.php)

Code:

```
@extends('layouts.app')

@section('content')
    <div class="row">
        <div class="col-lg-12 margin-tb">
            <div class="pull-left">
                <h2>Edit Product</h2>
            </div>
            <div class="pull-right">
                <a class="btn btn-primary" href="{{ route('products.index')
}}"> Back</a>
            </div>
        </div>
    </div>

    @if ($errors->any())
        <div class="alert alert-danger">
```

```
            <strong>Whoops!</strong> There were some problems with your inp
ut.<br><br>
            <ul>
                @foreach ($errors->all() as $error)
                    <li>{{ $error }}</li>
                @endforeach
            </ul>
        </div>
    @endif

    <form action="{{ route('products.update',$product->id) }}" method="POST
">
        @csrf
        @method('PUT')

        <div class="row">
            <div class="col-xs-12 col-sm-12 col-md-12">
                <div class="form-group">
                    <strong>Name:</strong>
                    <input type="text" name="name" value="{{ $product->name
}}" class="form-control" placeholder="Name">
                </div>
            </div>
            <div class="col-xs-12 col-sm-12 col-md-12">
                <div class="form-group">
                    <strong>Price:</strong>
                    <input type="number" step="0.01" name="price" value="{{
$product->price }}" class="form-control" placeholder="Price">
                </div>
            </div>
            <div class="col-xs-12 col-sm-12 col-md-12 text-center">
                <button type="submit" class="btn btn-primary">Submit</butto
n>
            </div>
        </div>
    </form>
@endsection
```

7. Routes သတ်မှတ်ခြင်း

routes/web.php ဖိုင်ကိုပြင်ဆင်ပါမယ်။

Code:

```
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\ProductController;

Route::resource('products', ProductController::class);

Route::get('/', function () {
    return redirect()->route('products.index');
});
```

8. Application ကို Run ခြင်း

Code:

```
php artisan serve
```

အခု browser မှာ http://localhost:8000 ကိုဖွင့်ကြည့်ရင် Product CRUD application ကိုအသုံးပြုနိုင်ပါပြီ။

UPLOAD FILE

Upload File

Upload File Features များ -

1. **Image Upload** - JPEG, PNG, JPG, GIF ဖိုင်များတင်နိုင်ခြင်း (အများဆုံး 2MB)
2. **File Information** - ဖိုင်အမည်၊ လမ်းကြောင်း၊ အရွယ်အစား သိမ်းဆည်းခြင်း
3. **Responsive Gallery** - မိုဘိုင်းနှင့် desktop နှစ်မျိုးလုံးအတွက် သင့်တော်သော gallery

✓ Project စတင်ခြင်း

Command:

```
composer create-project laravel/laravel image-upload
cd image-upload
```

✓ Database ချိတ်ဆက်ခြင်း

.env ဖိုင်ကိုပြင်ဆင်ပါမယ်။

Code:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=image_upload
DB_USERNAME=root
DB_PASSWORD=
```

✓ Photo Model နှင့် Migration ဖန်တီးခြင်း

Command:

```
php artisan make:model Photo -m
```

database/migrations/xxxx_create_photos_table.php ဖိုင်ကိုပြင်ဆင်ပါမယ်။

Code:

```
public function up()
{
    Schema::create('photos', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->string('path');
        $table->integer('size'); // bytes
        $table->timestamps();
    });
}
```

✓ Migration ကို run ပါမယ်။

Command:

```
php artisan migrate
```

✓ Photo Model ပြင်ဆင်ခြင်း

⇨ app/Models/Photo.php ဖိုင်ကိုပြင်ဆင်ပါမယ်။

Code:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Photo extends Model
{
    use HasFactory;

    protected $fillable = [
        'name',
        'path',
        'size'
    ];
}
```

✓ Photo Controller ဖန်တီးခြင်း

Command:

```
php artisan make:controller PhotoController --resource
```

⇨ app/Http/Controllers/PhotoController.php ဖိုင်ကိုပြင်ဆင်ပါမယ်။

Code:

```
<?php

namespace App\Http\Controllers;

use App\Models\Photo;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Storage;

class PhotoController extends Controller
{
    // Photo List
    public function index()
    {
        $photos = Photo::latest()->paginate(5);
        return view('photos.index', compact('photos'));
    }

    // Create Form
    public function create()
    {
        return view('photos.create');
    }

    // Store Photo
    public function store(Request $request)
    {
        $request->validate([
            'name' => 'required',
            'image' => 'required|image|mimes:jpeg,png,jpg,gif|max:2048'
        ]);

        $image = $request->file('image');
        $path = $image->store('photos', 'public');

        Photo::create([
```

```
            'name' => $request->name,
            'path' => $path,
            'size' => $image->getSize()
        ]);

        return redirect()->route('photos.index')
                         ->with('success', 'Photo uploaded successfully.');
    }

    // Show Photo
    public function show(Photo $photo)
    {
        return view('photos.show', compact('photo'));
    }

    // Delete Photo
    public function destroy(Photo $photo)
    {
        Storage::disk('public')->delete($photo->path);
        $photo->delete();

        return redirect()->route('photos.index')
                         ->with('success', 'Photo deleted successfully.');
    }
}
```

✓ Views ဖိုင်များဖန်တီးခြင်း

⇨ Layout File (resources/views/layouts/app.blade.php)

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Image Upload</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/boots
trap.min.css" rel="stylesheet">
</head>
<body>
    <div class="container mt-4">
        @yield('content')
    </div>
```

```
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/boots
trap.bundle.min.js"></script>
</body>
</html>
```

⇨ Index Page (resources/views/photos/index.blade.php)

Code:

```
@extends('layouts.app')

@section('content')
    <div class="d-flex justify-content-between align-items-center mb-4">
        <h1>Photo Gallery</h1>
        <a href="{{ route('photos.create') }}" class="btn btn-primary">Uplo
ad Photo</a>
    </div>

    @if ($message = Session::get('success'))
        <div class="alert alert-success">
            {{ $message }}
        </div>
    @endif

    <div class="row">
        @foreach ($photos as $photo)
        <div class="col-md-4 mb-4">
            <div class="card">
                <img src="{{ asset('storage/'.$photo->path) }}" class="card
-img-top" alt="{{ $photo->name }}">
                <div class="card-body">
                    <h5 class="card-title">{{ $photo->name }}</h5>
                    <p class="card-text">{{ number_format($photo->size / 10
24, 2) }} KB</p>
                    <form action="{{ route('photos.destroy',$photo->id) }}"
method="POST">
                        <a href="{{ route('photos.show',$photo->id) }}" cla
ss="btn btn-info">View</a>
                        @csrf
                        @method('DELETE')
                        <button type="submit" class="btn btn-danger">Delete
</button>
                    </form>
                </div>
            </div>
```

```
        </div>
        @endforeach
    </div>

    {!! $photos->links() !!}
@endsection
```

⇨ Create Form (resources/views/photos/create.blade.php)

Code:

```
@extends('layouts.app')

@section('content')
    <div class="row">
        <div class="col-lg-12 margin-tb">
            <div class="pull-left">
                <h2>Upload New Photo</h2>
            </div>
            <div class="pull-right">
                <a class="btn btn-primary" href="{{ route('photos.index') }
}"> Back</a>
            </div>
        </div>
    </div>

    @if ($errors->any())
        <div class="alert alert-danger">
            <strong>Whoops!</strong> There were some problems with your inp
ut.<br><br>
            <ul>
                @foreach ($errors->all() as $error)
                    <li>{{ $error }}</li>
                @endforeach
            </ul>
        </div>
    @endif

    <form action="{{ route('photos.store') }}" method="POST" enctype="multi
part/form-data">
        @csrf

        <div class="row">
            <div class="col-xs-12 col-sm-12 col-md-12">
```

```
                <div class="form-group">
                    <strong>Photo Name:</strong>
                    <input type="text" name="name" class="form-control" pla
ceholder="Name">
                </div>
            </div>
            <div class="col-xs-12 col-sm-12 col-md-12">
                <div class="form-group">
                    <strong>Image:</strong>
                    <input type="file" name="image" class="form-control">
                </div>
            </div>
            <div class="col-xs-12 col-sm-12 col-md-12 text-center">
                <button type="submit" class="btn btn-primary">Upload</butto
n>
            </div>
        </div>
    </form>
@endsection
```

⇨ Show Page (resources/views/photos/show.blade.php)

Code:

```
@extends('layouts.app')

@section('content')
  <div class="row">
    <div class="col-lg-12 margin-tb">
      <div class="pull-left">
        <h2>Photo Details</h2>
      </div>
      <div class="pull-right">
        <a class="btn btn-primary" href="{{ route('photos.index') }}"> Back</a>
      </div>
    </div>
  </div>

  <div class="row">
    <div class="col-md-6">
      <img src="{{ asset('storage/'.$photo->path) }}" class="img-fluid" alt="{{ $photo->name
}}">
    </div>
    <div class="col-md-6">
      <div class="form-group">
```

```
          <strong>Name:</strong>
          {{ $photo->name }}
        </div>
        <div class="form-group">
          <strong>File Path:</strong>
          {{ $photo->path }}
        </div>
        <div class="form-group">
          <strong>File Size:</strong>
          {{ number_format($photo->size / 1024, 2) }} KB
        </div>
        <div class="form-group">
          <strong>Uploaded At:</strong>
          {{ $photo->created_at }}
        </div>
      </div>
    </div>
@endsection
```

✓ Routes သတ်မှတ်ခြင်း

⇨ routes/web.php ဖိုင်ကိုပြင်ဆင်ပါမယ်။

Code:

```
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\PhotoController;

Route::resource('photos', PhotoController::class);

Route::get('/', function () {
    return redirect()->route('photos.index');
});
```

✓ Storage Link ချိတ်ဆက်ခြင်း

Command:

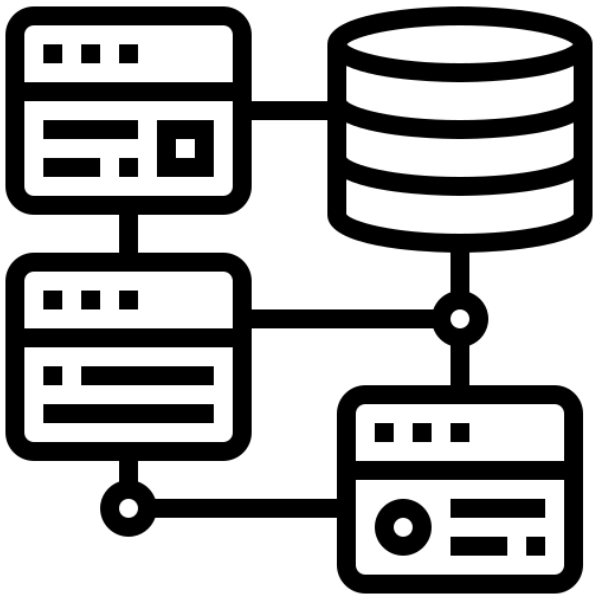
```
php artisan storage:link
```

✓ Application ကို Run ခြင်း

Command:

```
php artisan serve
```

MODEL RELATIONSHIPS



Model Relationships

Relationship တွေ မသုံးဘဲ Laravel app တွေကို ရေးနိုင်ပါတယ်။ Relationship တွေ မသုံးပဲ manual joins တွေ၊ multiple queries တွေနဲ့ ရေးနိုင်ပါတယ်။ ဒါပေမဲ့ Code ကို ရှုပ်ထွေးစေမယ် ၊ Performance ပြဿနာတွေ ဖြစ်စေမယ် ၊ Maintenance လုပ်ရ ခက်ခဲစေမယ် ၊ Laravel ရဲ့ built-in features တွေကို အပြည့်အဝ အသုံးမချနိုင်ဘဲ ဖြစ်သွားမယ် ၊ ဒါကြောင့် Database tables တွေကြားမှာ ဆက်နွယ်မှုရှိတဲ့အခါ Laravel relationships တွေကို အသုံးပြုဖို့ အကြံပြုပါတယ်။ အခြေခံအားဖြင့် Relationship ၃ မျိုးရှိပါတယ်။

- One to One Relationship
- One to Many Relationship
- Many to Many Relationship

ဆိုပြီးတော့ ဖြစ်ပါတယ်။ အသေးစိတ်ကို ဆက်ပြီးလေ့လာရအောင်။

Relationship အမျိုးအစားကို လိုက်ပြီး ဘယ်လို အသုံးပြုရမလဲဆိုတာကို ဆက်ပြီးပြောပြပေးပါမယ်။ ဥပမာ User တစ်ယောက်မှာ Profile တစ်ခုသာ ရှိတဲ့အခါ ၊ Product တစ်ခုမှာ ProductDetail တစ်ခုသာ ရှိတဲ့အခါ One To One Relationship ကိုအသုံးပြုရပါမယ်။ User တစ်ယောက်မှာ Post အများကြီး ရှိတဲ့အခါ ၊ Category တစ်ခုမှာ Product အများကြီး ရှိတဲ့အခါ One to Many Relationship ကို အသုံးပြုရပါမယ်။ Student အများကြီးမှာ Course အများကြီး တက်နေတဲ့အခါ ၊ Post အများကြီးမှာ Tag အများကြီး ရှိတဲ့အခါ မှာတော့ Many to Many Relationship ကိုအသုံးပြုရပါမယ်။

☑ One to One Relationship

One to One Relationship ဆိုတာက Model နှစ်ခုကြားမှာ တစ်ခုနဲ့တစ်ခု တိုက်ရိုက်ဆက်စပ်နေတဲ့ အချိုးအစားတစ်မျိုးဖြစ်ပါတယ်။ ဥပမာ:

- User တစ်ယောက်မှာ Phone တစ်လုံးပဲရှိနိုင်တယ်
- Phone တစ်လုံးမှာ User တစ်ယောက်ပဲရှိနိုင်တယ်

✓ Database Structure ပြင်ဆင်ခြင်း

⇨ Users Table Migration

Command:

```
php artisan make:migration create_users_table
```

Code:

```
Schema::create('users', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->string('email')->unique();
    $table->timestamps();
});
```

⇨ Phones Table Migration

Command:

```
php artisan make:migration create_phones_table
```

Code:

```
Schema::create('phones', function (Blueprint $table) {
    $table->id();
    $table->foreignId('user_id')->constrained()->onDelete('cascade');
    $table->string('number');
```

```
    $table->timestamps();
});
```

✓ Migration တွေကို run ပါမယ်:

Command:

```
php artisan migrate
```

✓ Model တွေဖန်တီးပြီး Relationship သတ်မှတ်ခြင်း

⇨ User Model

Command:

```
php artisan make:model User
```

⇨ app/Models/User.php မှာ:

Code:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    public function phone()
    {
        return $this->hasOne(Phone::class);
    }
}
```

⇨ Phone Model

Command:

```
php artisan make:model Phone
```

app/Models/Phone.php မှာ:

Code:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Phone extends Model
{
    public function user()
    {
        return $this->belongsTo(User::class);
    }
}
```

✓ Relationship အသုံးပြုနည်းများ

⇨ Data Create လုပ်ခြင်း

User နဲ့ Phone တစ်ခါတည်း Create လုပ်မယ်

Code:

```
$user = User::create([
    'name' => 'John Doe',
    'email' => 'john@example.com'
]);

$user->phone()->create([
    'number' => '09123456789'
]);
```

သီးသန့် Create လုပ်ပြီး Associate လုပ်မယ်

Code:

```
$user = User::create([
    'name' => 'Jane Doe',
    'email' => 'jane@example.com'
]);

$phone = Phone::create([
```

```
    'number' => '09987654321'
]);

$phone->user()->associate($user);
$phone->save();
```

⇨ Data Retrieve လုပ်ခြင်း

User ကနေ Phone ကိုရှာမယ်

Code:

```
$user = User::find(1);
$phone = $user->phone; // အဲ့မဟုတ် $user->phone()->first();
```

Phone ကနေ User ကိုရှာမယ်

Code:

```
$phone = Phone::find(1);
$user = $phone->user;
```

⇨ Data Update လုပ်ခြင်း

User ကနေ Phone Update လုပ်မယ်

Code:

```
$user = User::find(1);
$user->phone()->update(['number' => '09111111111']);
```

Phone ကနေ User ကို Update လုပ်မယ်

Code:

```
$phone = Phone::find(1);
$newUser = User::find(2);
$phone->user()->associate($newUser);
$phone->save();
```

⇨ Data Delete လုပ်ခြင်း

User ဖျက်လိုက်ရင် Phone ပါဖျက်မယ် (cascade)

Code:

```
$user = User::find(1);
$user->delete(); // Phone ပါအလိုအလျောက်ဖျက်သွားပါမယ်
```

Phone ပဲဖျက်မယ်

Code:

```
$user = User::find(1);
$user->phone()->delete();
```

✓ Eager Loading အသုံးပြုခြင်း

Relationship data တွေကို performance ကောင်းကောင်းနဲ့ ရယူနိုင်ဖို့ Eager Loading သုံးနိုင်ပါတယ်။

Code:

```
// User တွေအားလုံးကို သူတို့ရဲ့ Phone တွေနဲ့အတူ ရယူမယ်
$users = User::with('phone')->get();

foreach ($users as $user) {
    echo $user->phone->number;
}
```

✓ Relationship စစ်ဆေးခြင်း

⇨ Relationship ရှိမရှိစစ်ဆေးခြင်း

Code:

```
$user = User::find(1);

if ($user->phone) {
    // Phone ရှိတယ်
} else {
```

```
    // Phone မရှိဘူး
}
```

⇨ Relationship မရှိရင် Default Value ထည့်ခြင်း

Code:

```
$phoneNumber = $user->phone->number ?? 'No phone number';
```

✓ လက်တွေ့အသုံးဝင်မယ့် အခြားနမူနာများ

⇨ User နဲ့ Profile (One to One)

Code:

```
// User Model
public function profile()
{
    return $this->hasOne(Profile::class);
}

// Profile Model
public function user()
{
    return $this->belongsTo(User::class);
}
```

⇨ Product နဲ့ Inventory (One to One)

Code:

```
// Product Model
public function inventory()
{
    return $this->hasOne(Inventory::class);
}

// Inventory Model
public function product()
{
    return $this->belongsTo(Product::class);
}
```

✍ Short Notes:

- Foreign key ကို phone table မှာ user_id ဆိုပြီး ထည့်ပေးရပါမယ်
- hasOne နဲ့ belongsTo ကို မှန်မှန်ကန်ကန် သုံးဖို့လိုပါတယ်
- Eager Loading (with()) ကို N+1 query problem ကိုရှောင်ရှားဖို့သုံးသင့်ပါတယ်
- Cascade delete လုပ်ချင်ရင် migration မှာ onDelete('cascade') ထည့်ပေးရပါမယ်

One to One Relationship က Laravel Eloquent ORM ရဲ့ အခြေခံကျတဲ့ feature တစ်ခုဖြစ်ပြီး၊ မှန်ကန်စွာ အသုံးပြုနိုင်ရင် application ရဲ့ data structure ကို ပိုမိုကောင်းမွန်စွာ ဒီဇိုင်းဆွဲနိုင်မှာဖြစ်ပါတယ်။

☑ One to Many Relationship

One to Many Relationship ဆိုတာက Model တစ်ခုမှာ အခြား Model အများအပြားနဲ့ ဆက်စပ်နေတဲ့ အချိုးအစားတစ်မျိုးဖြစ်ပါတယ်။ ဥပမာ:

- User တစ်ယောက်မှာ Post အများအပြားရှိနိုင်တယ်
- Post တစ်ခုစီမှာ User တစ်ယောက်ပဲရှိနိုင်တယ်

✓ Database Structure ပြင်ဆင်ခြင်း

⇨ Users Table Migration

Command:

```
php artisan make:migration create_users_table
```

Code:

```
Schema::create('users', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->string('email')->unique();
    $table->timestamps();
});
```

⇨ Posts Table Migration

Command:

```
php artisan make:migration create_posts_table
```

Code:

```
Schema::create('posts', function (Blueprint $table) {
    $table->id();
    $table->foreignId('user_id')->constrained()->onDelete('cascade');
    $table->string('title');
    $table->text('content');
    $table->timestamps();
});
```

✓ Migration တွေကို run ပါမယ်:

Command:

```
php artisan migrate
```

✓ Model တွေဖန်တီးပြီး Relationship သတ်မှတ်ခြင်း

⇨ User Model

Code:

```
php artisan make:model User
```

app/Models/User.php မှာ:

Code:

```
<?php

namespace App\Models;
```

```
use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    public function posts()
    {
        return $this->hasMany(Post::class);
    }
}
```

⇨ Post Model

Code:

```
php artisan make:model Post
```

app/Models/Post.php မှာ:

Code:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
    public function user()
    {
        return $this->belongsTo(User::class);
    }
}
```

✓ Relationship အသုံးပြုနည်းများ

→ Data Create လုပ်ခြင်း

User နဲ့ Post တွေတစ်ခါတည်း Create လုပ်မယ်

Code:

```
$user = User::create([
    'name' => 'John Doe',
    'email' => 'john@example.com'
]);

$user->posts()->createMany([
    [
        'title' => 'First Post',
        'content' => 'Content of first post'
    ],
    [
        'title' => 'Second Post',
        'content' => 'Content of second post'
    ]
]);
```

သီးသန့် Create လုပ်ပြီး Associate လုပ်မယ်

Code:

```
$user = User::find(1);

$post = new Post([
    'title' => 'New Post',
    'content' => 'Post content here'
]);

$user->posts()->save($post);
```

→ Data Retrieve လုပ်ခြင်း

User ကနေ Post တွေကိုရှာမယ်

Code:

```
$user = User::find(1);
$posts = $user->posts; // ဒါမှမဟုတ် $user->posts()->get();
```

Post ကနေ User ကိုရှာမယ်

Code:

```
$post = Post::find(1);
$user = $post->user;
```

→ Data Update လုပ်ခြင်း

User ရဲ့ Post တွေကို Update လုပ်မယ်

Code:

```
$user = User::find(1);
$user->posts()->where('id', 1)->update(['title' => 'Updated Title']);
```

Post ရဲ့ User ကို Update လုပ်မယ်

Code:

```
$post = Post::find(1);
$newUser = User::find(2);
$post->user()->associate($newUser);
$post->save();
```

→ Data Delete လုပ်ခြင်း

User ဖျက်လိုက်ရင် Post တွေပါဖျက်မယ် (cascade)

Code:

```
$user = User::find(1);
$user->delete(); // Post တွေပါအလိုအလျောက်ဖျက်သွားပါမယ်
```

User ရဲ့ Post တစ်ခုပဲဖျက်မယ်

Code:

```
$user = User::find(1);
$user->posts()->where('id', 1)->delete();
```

✓ Eager Loading အသုံးပြုခြင်း

Code:

```
// User တွေအားလုံးကို သူတို့ရဲ့ Post တွေနဲ့အတူ ရယူမယ်
$users = User::with('posts')->get();

foreach ($users as $user) {
    foreach ($user->posts as $post) {
        echo $post->title;
    }
}
```

✓ Relationship စစ်ဆေးခြင်း

Post တွေရှိမရှိစစ်ဆေးခြင်း

Code:

```
$user = User::find(1);

if ($user->posts->isNotEmpty()) {
    // Post တွေရှိတယ်
} else {
    // Post မရှိဘူး
}
```

Post အရေအတွက်ရှာခြင်း

Code:

```
$postCount = $user->posts()->count();
```

✓ လက်တွေ့အသုံးဝင်မယ့် အခြားနမူနာများ

→ Category နဲ့ Product (One to Many)

Code:

```
// Category Model
public function products()
```

```
{
    return $this->hasMany(Product::class);
}

// Product Model
public function category()
{
    return $this->belongsTo(Category::class);
}
```

→ Country နဲ့ City (One to Many)

Code:

```
// Country Model
public function cities()
{
    return $this->hasMany(City::class);
}

// City Model
public function country()
{
    return $this->belongsTo(Country::class);
}
```

✓ အထူးသတိပြုရန် အချက်များ

- Foreign key ကို posts table မှာ user_id ဆိုပြီး ထည့်ပေးရပါမယ်
- hasMany နဲ့ belongsTo ကို မှန်မှန်ကန်ကန် သုံးဖို့လိုပါတယ်
- Eager Loading ကို N+1 query problem ကိုရှောင်ရှားဖို့သုံးသင့်ပါတယ်
- Cascade delete လုပ်ချင်ရင် migration မှာ onDelete('cascade') ထည့်ပေးရပါမယ်

One to Many Relationship က Laravel မှာ အသုံးများတဲ့ relationship တစ်မျိုးဖြစ်ပြီး၊ real-world application တွေမှာ အများဆုံးတွေ့ရတတ်ပါတယ်။ ဒီ relationship ကိုကောင်းစွာနားလည်ထားရင် data တွေကို ပိုမိုထိရောက်စွာ manage လုပ်နိုင်မှာဖြစ်ပါတယ်။

☑ Many to Many Relationship

Many to Many Relationship ဆိုတာက Model နှစ်ခုကြားမှာ အပြန်အလှန်ဆက်နွယ်မှုရှိတဲ့ အချိုးအစားတစ်မျိုးဖြစ်ပါတယ်။ ဥပမာ:

- Student တစ်ယောက်မှာ Course အများအပြားတက်နိုင်တယ်
- Course တစ်ခုမှာ Student အများအပြားရှိနိုင်တယ်

✓ Database Structure ပြင်ဆင်ခြင်း

Students Table Migration

Command:

```
php artisan make:migration create_students_table
```

Code:

```
Schema::create('students', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->timestamps();
});
```

Courses Table Migration

Code:

```
php artisan make:migration create_courses_table
```

Code:

```
Schema::create('courses', function (Blueprint $table) {
```

```
    $table->id();
    $table->string('title');
    $table->timestamps();
});
```

Pivot Table (course_student) Migration

Command:

```
php artisan make:migration create_course_student_table
```

Code:

```
Schema::create('course_student', function (Blueprint $table) {
    $table->foreignId('course_id')->constrained()->onDelete('cascade');
    $table->foreignId('student_id')->constrained()->onDelete('cascade');
    $table->primary(['course_id', 'student_id']);
    $table->timestamps(); // optional
});
```

Migration တွေကို run ပါမယ်:

Command:

```
php artisan migrate
```

✓ Model တွေဖန်တီးပြီး Relationship သတ်မှတ်ခြင်း

Student Model

Command:

```
php artisan make:model Student
```

app/Models/Student.php မှာ:

Code:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
```

```
class Student extends Model
{
    public function courses()
    {
        return $this->belongsToMany(Course::class);
    }
}
```

Course Model

Code:

```
php artisan make:model Course
```

app/Models/Course.php မှာ:

Code:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Course extends Model
{
    public function students()
    {
        return $this->belongsToMany(Student::class);
    }
}
```

✓ Relationship အသုံးပြုနည်းများ

⇨ Data Attach/Detach/Sync လုပ်ခြင်း

Student ကို Course နဲ့ချိတ်ဆက်မယ် (Attach)

Code:

```
$student = Student::find(1);
$student->courses()->attach([1, 2, 3]); // course_id တွေကို array အနေနဲ့ပေးမယ်
```

Student ကို Course ကနေဖြုတ်မယ် (Detach)

Code:

```
$student = Student::find(1);
$student->courses()->detach([2, 3]); // course_id 2 နဲ့3 ကိုဖြုတ်မယ်
```

Student ရဲ့ Courses တွေကို Sync လုပ်မယ်

Code:

```
$student = Student::find(1);
$student->courses()->sync([1, 4]); // ရှိပြီးသား courses တွေကိုဖြုတ်၊ အသစ် 1 နဲ့ 4 ကိုချိတ်မယ်
```

⇨ Data Retrieve လုပ်ခြင်း

Student ကနေ Courses တွေကိုရှာမယ်

Code:

```
$student = Student::find(1);
$courses = $student->courses;
```

Course ကနေ Students တွေကိုရှာမယ်

Code:

```
$course = Course::find(1);
$students = $course->students;
```

⇨ Pivot Table Data များကိုအသုံးပြုခြင်း

Pivot table မှာ အချက်အလက်တွေထပ်ထည့်မယ်

Code:

```
$student = Student::find(1);
$student->courses()->attach(1, ['enrolled_at' => now()]);
```

Pivot data တွေကိုဖတ်မယ်

Code:

```
$student = Student::find(1);
foreach ($student->courses as $course) {
    echo $course->pivot->enrolled_at;
}
```

⇨ Additional Condition များနဲ့ရှာဖွေခြင်း

Code:

```
// သင်တန်းတက်ထားတဲ့ ကျောင်းသားတွေကိုရှာမယ်
$course = Course::find(1);
$activeStudents = $course->students()
                         ->where('is_active', true)
                         ->get();
```

✓ Eager Loading အသုံးပြုခြင်း

Code:

```
// Student တွေအားလုံးကို သူတို့ရဲ့ Courses တွေနဲ့အတူ ရယူမယ်
$students = Student::with('courses')->get();

foreach ($students as $student) {
    foreach ($student->courses as $course) {
        echo $course->title;
    }
}
```

✓ Custom Pivot Model အသုံးပြုခြင်း

Pivot table ကို model အဖြစ်သတ်မှတ်နိုင်ပါတယ်။

Command:

```
php artisan make:model CourseStudent --pivot
```

app/Models/CourseStudent.php:

Code:

```
<?php

namespace App\Models;
```

```
use Illuminate\Database\Eloquent\Relations\Pivot;

class CourseStudent extends Pivot
{
    protected $casts = [
        'enrolled_at' => 'datetime',
    ];
}
```

Model တွေမှာ ပြင်ဆင်ပါမယ်:

Code:

```
// Student Model
public function courses()
{
    return $this->belongsToMany(Course::class)
                ->using(CourseStudent::class)
                ->withPivot('enrolled_at')
                ->withTimestamps();
}

// Course Model
public function students()
{
    return $this->belongsToMany(Student::class)
                ->using(CourseStudent::class)
                ->withPivot('enrolled_at')
                ->withTimestamps();
}
```

✓ လက်တွေ့အသုံးဝင်မယ့် အခြားနမူနာများ

⇨ User နဲ့ Role (Many to Many)

Code:

```
// User Model
public function roles()
{
    return $this->belongsToMany(Role::class);
}

// Role Model
public function users()
```

```
{
    return $this->belongsToMany(User::class);
}
```

⇨ Post နဲ့ Tag (Many to Many)

Code:

```
// Post Model
public function tags()
{
    return $this->belongsToMany(Tag::class);
}

// Tag Model
public function posts()
{
    return $this->belongsToMany(Post::class);
}
```

✓ အထူးသတိပြုရန် အချက်များ

1. Pivot table နာမည်ကို alphabetical order အတိုင်းပေးရပါမယ် (course_student, post_tag)
2. belongsToMany method ကိုနှစ်ဖက်စလုံးမှာသုံးရပါမယ်
3. Pivot table မှာ timestamps လိုချင်ရင် withTimestamps() method ကိုသုံးပါ
4. Pivot data တွေကိုလိုချင်ရင် withPivot() method ကိုသုံးပါ
5. Eager Loading ကို performance အတွက်အမြဲသုံးသင့်ပါတယ်

Many to Many Relationship က Laravel မှာအသုံးဝင်တဲ့ feature တစ်ခုဖြစ်ပြီး complex data relationships တွေကိုလွယ်ကူစွာကိုင်တွယ်နိုင်ပါတယ်။

Ajax

AJAX

AJAX (Asynchronous JavaScript and XML) ဆိုတာ စာမျက်နှာအသစ်တစ်ခုလုံး reload မလုပ်စေဘဲ server နဲ့ data တွေ ချိတ်ဆက်အသုံးပြုနိုင်တဲ့ technique တစ်ခုဖြစ်ပါတယ်။ Laravel တွင် AJAX ကို အဓိကအားဖြင့် jQuery သို့မဟုတ် Fetch API တို့နဲ့ အသုံးပြုကြပါတယ်။

အခြေခံ AJAX Request လုပ်ဆောင်နည်း

✓ jQuery အသုံးပြုပြီး AJAX Request ပို့နည်း

Code:

```
$.ajax({
    url: '/ajax-request',
    type: 'POST',
    data: {
        name: 'John Doe',
        age: 30
    },
    success: function(response) {
        console.log(response);
        // Server ကပြန်လာတဲ့ response ကို ကိုင်တွယ်မည်
    },
    error: function(xhr) {
        console.log(xhr.responseText);
        // Error ဖြစ်ပါက ကိုင်တွယ်မည်
    }
});
```

2. Fetch API အသုံးပြုပြီး AJAX Request ပို့နည်း

Code:

```
fetch('/ajax-request', {
    method: 'POST',
    headers: {
```

```
        'Content-Type': 'application/json',
        'X-CSRF-TOKEN': document.querySelector('meta[name="csrf-token"]').c
ontent
    },
    body: JSON.stringify({
        name: 'John Doe',
        age: 30
    })
})
.then(response => response.json())
.then(data => console.log(data))
.catch(error => console.error('Error:', error));
```

Laravel တွင် AJAX အသုံးပြုနည်း အဆင့်ဆင့်

✓ Route သတ်မှတ်ခြင်း

Code:

```
// web.php
Route::post('/products/ajax', [ProductController::class, 'ajaxRequest'])->n
ame('products.ajax');
```

2. Controller တွင် AJAX Request ကို ကိုင်တွယ်ခြင်း

Code:

```
namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Product;

class ProductController extends Controller
{
    public function ajaxRequest(Request $request)
    {
        if ($request->ajax()) {
            $products = Product::all();
            return response()->json($products);
        }

        return response()->json(['error' => 'Invalid request'], 400);
    }
```

```
}
```

3. Blade View တွင် AJAX Call ပြုလုပ်ခြင်း

Code:

```
<button id="loadProducts">Load Products</button>
<div id="productList"></div>

@section('scripts')
<script>
    $(document).ready(function() {
        $('#loadProducts').click(function() {
            $.ajax({
                url: "{{ route('products.ajax') }}",
                type: 'POST',
                dataType: 'json',
                headers: {
                    'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('cont
ent')
                },
                success: function(data) {
                    let html = '';
                    data.forEach(product => {
                        html += `<div>${product.name} - $${product.price}</
div>`;
                    });
                    $('#productList').html(html);
                },
                error: function(xhr) {
                    console.log(xhr.responseText);
                }
            });
        });
    });
</script>
@endsection
```

✓ AJAX တွင် CSRF Protection

Laravel တွင် AJAX request တွေအတွက် CSRF token ထည့်ပေးဖို့လိုပါတယ်။

1. Meta Tag ထည့်သွင်းခြင်း

Code:

```
<meta name="csrf-token" content="{{ csrf_token() }}">
```

2. AJAX Setup တွင် Global CSRF Token ထည့်ခြင်း

Code:

```
$.ajaxSetup({
    headers: {
        'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')
    }
});
```

အသုံးဝင်သော AJAX Pattern များ

1. Form Submit with AJAX

Code:

```
$('#productForm').submit(function(e) {
    e.preventDefault();

    $.ajax({
        url: $(this).attr('action'),
        type: 'POST',
        data: $(this).serialize(),
        success: function(response) {
            alert('Product saved successfully!');
        },
        error: function(xhr) {
            console.log(xhr.responseText);
        }
    });
});
```

2. Infinite Scroll with AJAX

Code:

```
$(window).scroll(function() {
    if ($(window).scrollTop() + $(window).height() >= $(document).height()
- 100) {
        loadMoreProducts();
    }
});

function loadMoreProducts(page = 1) {
    $.ajax({
        url: '/products/load-more?page=' + page,
        type: 'GET',
        success: function(response) {
            $('#productContainer').append(response.html);
        }
    });
}
```

3. Live Search with AJAX

Code:

```
$('#searchInput').keyup(function() {
    let query = $(this).val();

    $.ajax({
        url: '/products/search',
        type: 'GET',
        data: {query: query},
        success: function(response) {
            $('#searchResults').html(response);
        }
    });
});
```

✓ AJAX Error Handling

AJAX request တွေမှာ error တွေကို ကောင်းမွန်စွာ handle လုပ်ဖို့လိုပါတယ်။

Code:

```
$.ajax({
    url: '/some-url',
    type: 'POST',
    data: {...},
    statusCode: {
        404: function() {
            alert('Page not found');
        },
        500: function() {
            alert('Server error');
        },
        422: function(xhr) {
            let errors = xhr.responseJSON.errors;
            // Form validation errors များကို ပြသမည်
        }
    }
});
```

✓ Laravel AJAX Best Practices

1. **Route naming** - AJAX routes တွေကို နာမည်ပေးတဲ့အခါ `.ajax` သို့မဟုတ် `.api` suffix ထည့်ပေးပါ

   Code:

   ```
   Route::post('/products/ajax', 'ProductController@ajax')->name('products.ajax');
   ```

2. **Response standardization** - AJAX responses တွေကို standard format နဲ့ပြန်ပေးပါ

   Code:

```
return response()->json([
    'success' => true,
    'data' => $products,
    'message' => 'Products loaded successfully'
]);
```

3. **Validation** - AJAX request တွေအတွက် validation လုပ်ပါ

Code:

```
$validated = $request->validate([
    'name' => 'required|string|max:255',
    'price' => 'required|numeric'
]);
```

4. **Loading indicators** - AJAX request တွေလုပ်နေစဉ် loading indicators ပြသပါ

Code:

```
$(document).ajaxStart(function() {
    $('#loading').show();
}).ajaxStop(function() {
    $('#loading').hide();
});
```

လေ့ကျင့်ရန် -

Item Management System

1. Project Setup

Database Migration ဖန်တီးခြင်း

Code:

```
// database/migrations/xxxx_xx_xx_create_items_table.php
Schema::create('items', function (Blueprint $table) {
    $table->id();
    $table->string('category');
    $table->string('name');
    $table->decimal('price', 8, 2);
```

```
    $table->string('image')->nullable();
    $table->timestamps();
});
```

Item Model ဖန်တီးခြင်း

Code:

```
// app/Models/Item.php
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Item extends Model
{
    use HasFactory;

    protected $fillable = ['category', 'name', 'price', 'image'];
}
```

2. Route များ သတ်မှတ်ခြင်း

Code:

```
// routes/web.php
use App\Http\Controllers\ItemController;

Route::get('/items', [ItemController::class, 'index'])->name('items.index')
;
Route::post('/items', [ItemController::class, 'store'])->name('items.store'
);
Route::get('/items/{item}', [ItemController::class, 'show'])->name('items.s
how');
Route::put('/items/{item}', [ItemController::class, 'update'])->name('items
.update');
Route::delete('/items/{item}', [ItemController::class, 'destroy'])->name('i
tems.destroy');
```

3. ItemController ဖန်တီးခြင်း

Code:

```
// app/Http/Controllers/ItemController.php
namespace App\Http\Controllers;

use App\Models\Item;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Storage;

class ItemController extends Controller
{
    public function index()
    {
        $items = Item::all();
        return view('items.index', compact('items'));
    }

    public function store(Request $request)
    {
        $validated = $request->validate([
            'category' => 'required|string|max:255',
            'name' => 'required|string|max:255',
            'price' => 'required|numeric',
            'image' => 'nullable|image|mimes:jpeg,png,jpg,gif|max:2048'
        ]);

        if ($request->hasFile('image')) {
            $validated['image'] = $request->file('image')->store('items', '
public');
        }

        $item = Item::create($validated);

        return response()->json([
            'success' => true,
            'item' => $item,
            'message' => 'Item created successfully'
        ]);
    }

    public function show(Item $item)
    {
        return response()->json($item);
    }

    public function update(Request $request, Item $item)
    {
        $validated = $request->validate([
```

```
            'category' => 'required|string|max:255',
            'name' => 'required|string|max:255',
            'price' => 'required|numeric',
            'image' => 'nullable|image|mimes:jpeg,png,jpg,gif|max:2048'
        ]);

        if ($request->hasFile('image')) {
            // Delete old image if exists
            if ($item->image) {
                Storage::disk('public')->delete($item->image);
            }
            $validated['image'] = $request->file('image')->store('items', '
public');
        }

        $item->update($validated);

        return response()->json([
            'success' => true,
            'item' => $item,
            'message' => 'Item updated successfully'
        ]);
    }

    public function destroy(Item $item)
    {
        if ($item->image) {
            Storage::disk('public')->delete($item->image);
        }

        $item->delete();

        return response()->json([
            'success' => true,
            'message' => 'Item deleted successfully'
        ]);
    }
}
```

4. Blade View ဖန်တီးခြင်း

Code:

```
<!-- resources/views/items/index.blade.php -->
@extends('layouts.app')
```

```
@section('content')
<div class="container">
    <h1>Item Management</h1>

    <!-- Create Item Form -->
    <div class="card mb-4">
        <div class="card-header">
            <h4>Add New Item</h4>
        </div>
        <div class="card-body">
            <form id="itemForm">
                @csrf
                <input type="hidden" id="itemId" name="id">

                <div class="row">
                    <div class="col-md-4">
                        <div class="form-group">
                            <label for="category">Category</label>
                            <input type="text" class="form-control" id="cat
egory" name="category" required>
                        </div>
                    </div>
                    <div class="col-md-4">
                        <div class="form-group">
                            <label for="name">Name</label>
                            <input type="text" class="form-control" id="nam
e" name="name" required>
                        </div>
                    </div>
                    <div class="col-md-4">
                        <div class="form-group">
                            <label for="price">Price</label>
                            <input type="number" step="0.01" class="form-co
ntrol" id="price" name="price" required>
                        </div>
                    </div>
                </div>

                <div class="form-group">
                    <label for="image">Image</label>
                    <input type="file" class="form-control-file" id="image"
name="image">
                    <div id="imagePreview" class="mt-2"></div>
                </div>
```

```
                <button type="submit" class="btn btn-primary" id="saveBtn">
Save Item</button>
                <button type="button" class="btn btn-secondary" id="cancelB
tn" style="display:none;">Cancel</button>
            </form>
        </div>
    </div>

    <!-- Items Table -->
    <div class="card">
        <div class="card-header">
            <h4>Item List</h4>
        </div>
        <div class="card-body">
            <div class="table-responsive">
                <table class="table table-bordered" id="itemsTable">
                    <thead>
                        <tr>
                            <th>ID</th>
                            <th>Category</th>
                            <th>Name</th>
                            <th>Price</th>
                            <th>Image</th>
                            <th>Actions</th>
                        </tr>
                    </thead>
                    <tbody>
                        <!-- Items will be loaded via AJAX -->
                    </tbody>
                </table>
            </div>
        </div>
    </div>
</div>
@endsection

@section('scripts')
<script>
$(document).ready(function() {
    // CSRF Token Setup
    $.ajaxSetup({
        headers: {
            'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')
        }
    });
```

```
    // Load all items
    function loadItems() {
        $.ajax({
            url: "{{ route('items.index') }}",
            type: 'GET',
            success: function(response) {
                let html = '';
                response.forEach(item => {
                    html += `
                    <tr id="item_${item.id}">
                        <td>${item.id}</td>
                        <td>${item.category}</td>
                        <td>${item.name}</td>
                        <td>$${item.price.toFixed(2)}</td>
                        <td>
                            ${item.image ? `<img src="/storage/${item.image
}" width="50" height="50">` : 'No Image'}
                        </td>
                        <td>
                            <button class="btn btn-sm btn-warning editBtn"
data-id="${item.id}">Edit</button>
                            <button class="btn btn-sm btn-danger deleteBtn"
data-id="${item.id}">Delete</button>
                        </td>
                    </tr>`;
                });
                $('#itemsTable tbody').html(html);
            }
        });
    }

    // Initial load
    loadItems();

    // Create or Update Item
    $('#itemForm').submit(function(e) {
        e.preventDefault();

        let formData = new FormData(this);
        let itemId = $('#itemId').val();
        let url = itemId ? `/items/${itemId}` : "{{ route('items.store') }}
";
        let method = itemId ? 'PUT' : 'POST';

        $.ajax({
            url: url,
```

```
            type: method,
            data: formData,
            contentType: false,
            processData: false,
            success: function(response) {
                $('#itemForm')[0].reset();
                $('#imagePreview').html('');
                $('#itemId').val('');
                $('#saveBtn').text('Save Item');
                $('#cancelBtn').hide();
                loadItems();
                alert(response.message);
            },
            error: function(xhr) {
                console.log(xhr.responseText);
                alert('Error occurred. Please try again.');
            }
        });
    });

    // Edit Item
    $(document).on('click', '.editBtn', function() {
        let itemId = $(this).data('id');

        $.ajax({
            url: `/items/${itemId}`,
            type: 'GET',
            success: function(response) {
                $('#itemId').val(response.id);
                $('#category').val(response.category);
                $('#name').val(response.name);
                $('#price').val(response.price);

                if (response.image) {
                    $('#imagePreview').html(`<img src="/storage/${response.
image}" width="100">`);
                }

                $('#saveBtn').text('Update Item');
                $('#cancelBtn').show();
            }
        });
    });

    // Cancel Edit
    $('#cancelBtn').click(function() {
```

```
        $('#itemForm')[0].reset();
        $('#imagePreview').html('');
        $('#itemId').val('');
        $('#saveBtn').text('Save Item');
        $(this).hide();
    });

    // Delete Item
    $(document).on('click', '.deleteBtn', function() {
        if (confirm('Are you sure you want to delete this item?')) {
            let itemId = $(this).data('id');

            $.ajax({
                url: `/items/${itemId}`,
                type: 'DELETE',
                success: function(response) {
                    loadItems();
                    alert(response.message);
                }
            });
        }
    });

    // Image Preview
    $('#image').change(function() {
        let reader = new FileReader();
        reader.onload = (e) => {
            $('#imagePreview').html(`<img src="${e.target.result}" width="1
00">`);
        }
        reader.readAsDataURL(this.files[0]);
    });
});
</script>
@endsection
```

✓ Short Notes:

1. File Upload Handling:
    - FormData object ကိုသုံးပြီး file upload လုပ်ပါတယ်

- contentType: false နဲ့ processData: false ကို AJAX request တွင် ထည့်ပေးရပါမယ်

2. Image Management:
   - Image တွေကို storage/app/public/items folder ထဲမှာ သိမ်းပါတယ်
   - php artisan storage:link command ကို run ပြီး public storage link လုပ်ပေးရပါမယ်

3. CRUD Operations:
   - Create: POST /items
   - Read: GET /items (list) and GET /items/{id} (single item)
   - Update: PUT /items/{id}
   - Delete: DELETE /items/{id}
4. Responsive Design:
   - Bootstrap ကိုသုံးပြီး mobile friendly ဖြစ်အောင် design လုပ်ထားပါတယ်
5. Error Handling:
   - AJAX request တွေမှာ error တွေကို ကောင်းမွန်စွာ handle လုပ်ထားပါတယ်

✓ အသုံးပြုရန် အဆင့်ဆင့်

1. Database migration နဲ့ model ကို create လုပ်ပါ

   Command:

```
php artisan make:model Item -m
```

2. Migration file ကို edit လုပ်ပြီး up() method ကို update လုပ်ပါ
3. Model ကို edit လုပ်ပြီး $fillable property ကို set လုပ်ပါ
4. Controller create လုပ်ပါ

   Command:

```
php artisan make:controller ItemController
```

5. Route တွေကို web.php မှာ define လုပ်ပါ
6. View file ကို create လုပ်ပါ (resources/views/items/index.blade.php)
7. Storage link create လုပ်ပါ

   Command:

```
php artisan storage:link
```

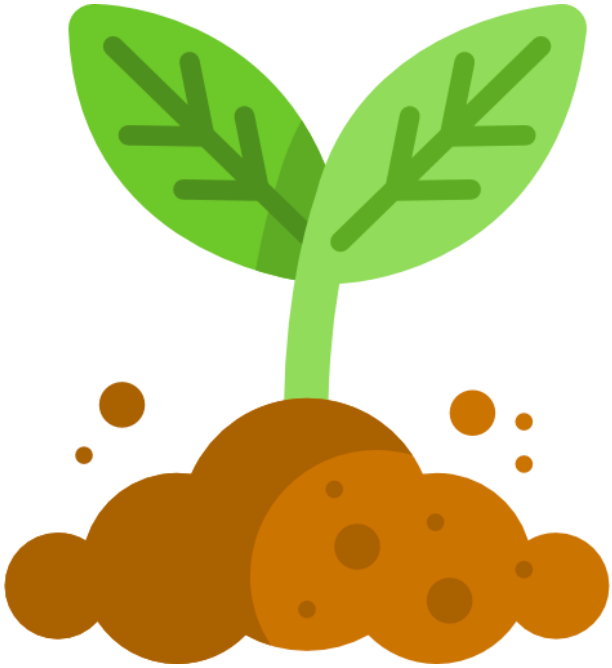
8. JavaScript dependencies (jQuery) ကို include လုပ်ပါ
   - Laravel UI သို့မဟုတ် CDN ကနေ include လုပ်နိုင်ပါတယ်
9. Application ကို run ပါ

   Command:

```
php artisan serve
```

ဒီ application မှာ AJAX ကိုသုံးပြီး page refresh မလုပ်စေဘဲ CRUD operations တွေကို လုပ်ဆောင်နိုင်ပါတယ်။ Image upload, form validation, error handling တွေပါ အပြည့်အစုံပါဝင်ပါတယ်။

Database Seeding



Database Seeding

Laravel မှာ Database Seeding ဆိုသည်မှာ အချက်အလက်များကို အလိုအလျောက်ထည့်သွင်းခြင်း (Auto-populate) ဖြစ်ပါတယ်။ Application Testing အတွက် Fake Data တွေထည့်မယ်။ Application project startup မှာ Default Users, Categories စသည့် Data တွေ ထည့်သွင်းပြီးတော့ project ကို မြန်မြန်ဆန်ဆန် ပြီးအောင် ဆောက်မယ်။ အဲဒါကြောင့် Application Testing, Demo Apps နဲ့ Development အတွက် အလွန်အသုံးဝင်ပါတယ်။ Faker, Factories နှင့် Manual Data များကို လွယ်ကူစွာ အသုံးပြုနိုင်ပြီး Database ကို Auto-fill လုပ်နိုင်ပါတယ်။

✓ Seeder ဖန်တီးခြင်း

Laravel တွင် Seeder ဖန်တီးရန် artisan command ကို အသုံးပြုနိုင်ပါတယ်။

Command:

```
php artisan make:seeder UsersTableSeeder
```

ဒီ command က database/seeders ဖိုလ်ဒါအတွင်း UsersTableSeeder.php ဖိုင်ကို ဖန်တီးပေးပါမယ်။

✓ Seeder အတွက် Data ထည့်သွင်းခြင်း

run() method အတွင်း **Faker** (သို့) **Manual Data** ထည့်နိုင်ပါတယ်။

ဥပမာ (Faker အသုံးပြု၍ User Data ထည့်ခြင်း)

Code:

```
<?php
```

```
namespace Database\Seeders;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Hash;
use Faker\Factory as Faker;

class UsersTableSeeder extends Seeder
{
    public function run()
    {
        $faker = Faker::create();

        foreach (range(1, 20) as $index) {
            DB::table('users')->insert([
                'name' => $faker->name,
                'email' => $faker->unique()->safeEmail,
                'password' => Hash::make('password'),
                'created_at' => now(),
                'updated_at' => now(),
            ]);
        }
    }
}
```

ဥပမာ (Manual Data ထည့်ခြင်း)

Code:

```
public function run()
{
    DB::table('users')->insert([
        [
            'name' => 'Admin',
            'email' => 'admin@example.com',
            'password' => Hash::make('admin123'),
            'created_at' => now(),
            'updated_at' => now(),
        ],
        [
            'name' => 'User',
            'email' => 'user@example.com',
            'password' => Hash::make('user123'),
            'created_at' => now(),
            'updated_at' => now(),
        ],
```

```
    ]);
}
```

✓ DatabaseSeeder.php တွင် Seeder ကို Call ခြင်း

အဓိက DatabaseSeeder.php ဖိုင်တွင် သင်၏ Seeder ကို ခေါ်ရုပါမယ်။

Code:

```
<?php
namespace Database\Seeders;
use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    public function run()
    {
        $this->call([
            UsersTableSeeder::class,
            // အခြား Seeder များကိုလည်း ဒီမှာ ထည့်နိုင်ပါတယ်
        ]);
    }
}
```

✓ Seeder ကို Run ခြင်း

Seeder ကို Run ရန် အောက်ပါ Command များကို အသုံးပြုနိုင်ပါတယ်။

⇨ တစ်ခုတည်းသော Seeder Run ခြင်း

Command:

```
php artisan db:seed --class=UsersTableSeeder
```

⇨ DatabaseSeeder.php မှတစ်ဆင့် အားလုံးကို Run ခြင်း

Code:

```
php artisan db:seed
```

⇨ Migration + Seeding အားလုံးကို တစ်ခါတည်း Run ခြင်း

Code:

```
php artisan migrate:fresh --seed
```

(⚠ `migrate:fresh` က **အရင်ရှိပြီးသား Tables များကို Drop** လုပ်ပြီး ပြန်လည်ဖန်တီးပါမယ်။)

✓ Model Factories အသုံးပြုခြင်း (ပိုမိုကောင်းမွန်သောနည်း)

Laravel တွင် **Factory** များကို အသုံးပြု၍ **အဆင့်မြင့် Seeding** လုပ်နိုင်ပါတယ်။

⇨ Factory ဖန်တီးခြင်း

Command:

```
php artisan make:factory UserFactory --model=User
```

⇨ Factory တွင် Fake Data သတ်မှတ်ခြင်း

Code:

```
<?php
namespace Database\Factories;
use Illuminate\Database\Eloquent\Factories\Factory;
use Illuminate\Support\Facades\Hash;

class UserFactory extends Factory
{
    public function definition()
    {
        return [
            'name' => $this->faker->name(),
            'email' => $this->faker->unique()->safeEmail(),
            'password' => Hash::make('password'),
        ];
    }
}
```

⇨ Seeder တွင် Factory အသုံးပြုခြင်း

Code:

```
use App\Models\User;

public function run()
{
    User::factory()->count(50)->create(); // User 50 ယောက်ကို Auto-generate လုပ်မည်
}
```

📌 Short Notes:

- ✅ Faker ကို အသုံးပြု၍ Realistic Fake Data များ ထည့်နိုင်ပါတယ်။
- ✅ Relationships (User-Post Comments) များကိုလည်း Seeding လုပ်နိုင်ပါတယ်။
- ✅ Production မှာ မဖြစ်မနေ php artisan db:seed မလုပ်ပါနှင့် (အချက်အလက်များ ပျက်စီးနိုင်)

💻 လေ့ကျင့်ရန်

ဒီ Laravel Database Seeding Exercise မှာ User နဲ့ Product CRUD operations အတွက် အခြေခံ structure တစ်ခုကို ဖော်ပြထားပါတယ်။ Faker နဲ့ Factory pattern များကို အသုံးပြု၍ realistic fake data များ generate လုပ်ကြမှာ ဖြစ်ပါတယ်။ Database seeding ကို အသုံးပြုခြင်းအားဖြင့် development နဲ့ testing phases များတွင် အချိန်ကုန်သက်သာစေမှာ ဖြစ်ပါတယ်။

✓ Project Structure

```
app/
├─── Models/
|    ├─── User.php
|    └─── Product.php
database/
├─── factories/
|    ├─── UserFactory.php
|    └─── ProductFactory.php
├─── migrations/
|    ├─── 2014_10_12_000000_create_users_table.php
|    ├─── 2014_10_12_100000_create_password_resets_table.php
|    └─── 2023_01_01_000000_create_products_table.php
├─── seeders/
|    ├─── DatabaseSeeder.php
|    ├─── UserSeeder.php
|    └─── ProductSeeder.php
```

✓ Models & Migrations

⇨ User Model (app/Models/User.php)

Code:

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;

class User extends Authenticatable
{
    use HasFactory, Notifiable;

    protected $fillable = [
        'name',
        'email',
        'password',
        'userType',
        'image'
    ];

    protected $hidden = [
```

```
        'password',
        'remember_token',
    ];
}
```

⇨ Product Model (app/Models/Product.php)

Code:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Product extends Model
{
    use HasFactory;

    protected $fillable = [
        'category',
        'name',
        'price',
        'image'
    ];
}
```

⇨ Create Users Table Migration

Code:

```
Schema::create('users', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->string('email')->unique();
    $table->string('password');
    $table->enum('userType', ['admin', 'user'])->default('user');
    $table->string('image')->nullable();
    $table->rememberToken();
    $table->timestamps();
});
```

⇨ Create Products Table Migration

Code:

```
Schema::create('products', function (Blueprint $table) {
    $table->id();
    $table->string('category');
    $table->string('name');
    $table->decimal('price', 8, 2);
    $table->string('image')->nullable();
    $table->timestamps();
});
```

✓ Factories

⇨ User Factory (database/factories/UserFactory.php)

Code:

```
<?php

namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Str;

class UserFactory extends Factory
{
    public function definition()
    {
        return [
            'name' => $this->faker->name(),
            'email' => $this->faker->unique()->safeEmail(),
            'password' => Hash::make('password'),
            'userType' => $this->faker->randomElement(['admin', 'user']),
            'image' => $this->faker->imageUrl(100, 100, 'people'),
            'remember_token' => Str::random(10),
        ];
    }
}
```

⇨ Product Factory (database/factories/ProductFactory.php)

Code:

```
<?php

namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;

class ProductFactory extends Factory
{
    public function definition()
    {
        return [
            'category' => $this->faker->randomElement(['Electronics', 'Clot
hing', 'Food', 'Furniture']),
            'name' => $this->faker->word(),
            'price' => $this->faker->randomFloat(2, 10, 1000),
            'image' => $this->faker->imageUrl(400, 400, 'products'),
        ];
    }
}
```

✓ Seeders

⇨ User Seeder (database/seeders/UserSeeder.php)

Code:

```
<?php

namespace Database\Seeders;

use App\Models\User;
use Illuminate\Database\Seeder;

class UserSeeder extends Seeder
{
    public function run()
    {
        // Create admin user
        User::create([
            'name' => 'Admin User',
            'email' => 'admin@example.com',
            'password' => bcrypt('admin123'),
```

```
            'userType' => 'admin',
            'image' => 'https://i.pravatar.cc/150?img=1'
        ]);

        // Create regular users using factory
        User::factory()->count(10)->create();
    }
}
```

⇨ Product Seeder (database/seeders/ProductSeeder.php)

Code:

```
<?php

namespace Database\Seeders;

use App\Models\Product;
use Illuminate\Database\Seeder;

class ProductSeeder extends Seeder
{
    public function run()
    {
        // Create sample products
        Product::create([
            'category' => 'Electronics',
            'name' => 'Smartphone',
            'price' => 599.99,
            'image' => 'https://via.placeholder.com/400x400?text=Smartphone
'
        ]);

        Product::create([
            'category' => 'Clothing',
            'name' => 'T-Shirt',
            'price' => 19.99,
            'image' => 'https://via.placeholder.com/400x400?text=T-Shirt'
        ]);

        // Generate random products
        Product::factory()->count(20)->create();
    }
}
```

⇨ Database Seeder (database/seeders/DatabaseSeeder.php)

Code:

```
<?php

namespace Database\Seeders;

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    public function run()
    {
        $this->call([
            UserSeeder::class,
            ProductSeeder::class,
        ]);
    }
}
```

✓ Running the Seeders

Run the following command to migrate and seed your database:

Command:

```
php artisan migrate:fresh --seed
```

✓ Testing Results

⇨ Expected Database Contents

Users Table:

- 1 admin user (admin@example.com)
- 10 random users generated by factory

Products Table:

- 2 manually created products (Smartphone and T-Shirt)
- 20 random products generated by factory

✓ Verification Commands

1. Check users count:

Command:

```
php artisan tinker
>>> \App\Models\User::count();
# Should return 11 (1 admin + 10 users)
```

2. Check products count:

Command:

```
php artisan tinker
>>> \App\Models\Product::count();
# Should return 22 (2 manual + 20 factory)
```

3. Verify admin user exists:

Command:

```
php artisan tinker
>>> \App\Models\User::where('email', 'admin@example.com')->first();
# Should return the admin user record
```

✓ CRUD Operations

⇨ User CRUD Example

Code:

```
// Create
$user = User::create([
    'name' => 'New User',
    'email' => 'new@example.com',
    'password' => bcrypt('password'),
    'userType' => 'user'
]);

// Read
$user = User::find(1);

// Update
$user->update(['name' => 'Updated Name']);

// Delete
```

```
$user->delete();
```

⇨ Product CRUD Example

Code:

```
// Create
$product = Product::create([
    'category' => 'Electronics',
    'name' => 'Laptop',
    'price' => 999.99
]);

// Read
$product = Product::find(1);

// Update
$product->update(['price' => 899.99]);

// Delete
$product->delete();
```

Data Table

Data Table

Data Table ဆိုတာ web application တွေမှာ ဇယားပုံစံနဲ့ဒေတာတွေကို စနစ်တကျပြသပေးတဲ့ UI component တစ်ခုဖြစ်ပါတယ်။ ဒေတာတွေကို အောက်ပါ feature တွေနဲ့ ပြသပေးပါတယ်:

1. Sorting - Column တစ်ခုချင်းစီကို နှိပ်ပြီး ascending/descending စီနိုင်ပါတယ်
2. Pagination - ဒေတာများလာရင် page တွေခွဲပြီး ပြပေးပါတယ်
3. Searching - ရှာလို့ရတဲ့ search box ပါဝင်ပါတယ်
4. Filtering - သီးသန့်စစ်ထုတ်ချင်တဲ့ဒေတာတွေကို စစ်နိုင်ပါတယ်

✓ Data Table ၏ အရေးပါမှု

1. ဒေတာကြီးမားမှုကို စီမံနိုင်ခြင်း
    - ဒေတာအများကြီးကို တစ်ပြိုင်နက်တည်း ဖတ်ရှုရလွယ်ကူစေပါတယ်
2. User Experience ကောင်းမွန်ခြင်း
    - လိုအပ်တဲ့ဒေတာကို အလွယ်တကူရှာဖွေနိုင်ပါတယ်
3. အချိန်ကုန်သက်သာခြင်း
    - Server-side processing နဲ့ဆိုရင် ဒေတာအားလုံးကို တစ်ခါတည်းမလိုဘဲ အလုပ်လုပ်နိုင်ပါတယ်

✓ Data Table အမျိုးအစားများ

1. Client-side Data Table
    - ဒေတာအားလုံးကို front-end မှာ process လုပ်ပါတယ်
    - ဥပမာ: jQuery DataTables (client-side mode)
2. Server-side Data Table
    - ဒေတာတွေကို server မှာ process လုပ်ပြီး လိုအပ်တဲ့အပိုင်းကိုသာ ပြန်ပို့ပါတယ်
    - ဥပမာ: Laravel Yajra DataTables

✓ Laravel တွင် Data Table အသုံးပြုနည်း

Code:

```
// Controller
public function index()
{
    return DataTables::of(User::query())->make(true);
}

// Blade View
<table id="users-table" class="table table-bordered">
    <thead>
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Email</th>
        </tr>
    </thead>
</table>

<script>
$(document).ready(function() {
    $('#users-table').DataTable({
        processing: true,
        serverSide: true,
        ajax: '{{ route('users.data') }}',
        columns: [
            { data: 'id', name: 'id' },
            { data: 'name', name: 'name' },
            { data: 'email', name: 'email' }
        ]
    });
});
</script>
```

✓ Data Table ၏ အားသာချက်များ

1. ဒေတာများကို စနစ်တကျ စီစဉ်ပြသနိုင်ခြင်း
2. လိုအပ်တဲ့ feature တွေ built-in ပါဝင်နေခြင်း
3. Responsive design နဲ့ mobile-friendly ဖြစ်ခြင်း

4. Customization လုပ်ရလွယ်ကူခြင်း

Data Table ကို e-commerce site တွေ, admin panel တွေ, report တွေပြတဲ့နေရာတွေမှာ အများဆုံးတွေ့ရပါတယ်။

💻 လေ့ကျင့်ရန် -

ဒီ application မှာ Laravel နဲ့ DataTable ကို အသုံးပြုပြီးတော့ search, sort, filter တွေပါတဲ့ professional-grade data table တစ်ခုကို တည်ဆောက်ကြပါမယ်။ Database မှာ item 20 ခုရှိပြီး အထက်ပါ features တွေအားလုံးပါဝင်တဲ့ application တစ်ခုဖြစ်ရပါမယ်။

Project Setup

✓ Database Migration ဖန်တီးခြင်း

Code:

```
// database/migrations/xxxx_xx_xx_create_items_table.php
Schema::create('items', function (Blueprint $table) {
    $table->id();
    $table->string('category');
    $table->string('name');
    $table->decimal('price', 8, 2);
    $table->string('image')->nullable();
    $table->timestamps();
});
```

✓ Item Model ဖန်တီးခြင်း

Code:

```
// app/Models/Item.php
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Item extends Model
```

```
{
    use HasFactory;

    protected $fillable = ['category', 'name', 'price', 'image'];
}
```

✓ Seeder ဖန်တီးခြင်း

Code:

```
// database/seeders/ItemSeeder.php
public function run()
{
    $categories = ['Electronics', 'Clothing', 'Food', 'Books', 'Furniture']
;

    \App\Models\Item::factory()->count(20)->create([
        'category' => function() use ($categories) {
            return $categories[array_rand($categories)];
        },
        'price' => rand(1000, 100000) / 100
    ]);
}
```

✓ Route များ သတ်မှတ်ခြင်း

Code:

```
// routes/web.php
use App\Http\Controllers\ItemController;

Route::get('/items', [ItemController::class, 'index'])->name('items.index')
;
Route::get('/items/data', [ItemController::class, 'getData'])->name('items.
data');
```

✓ ItemController ဖန်တီးခြင်း

Code:

```
// app/Http/Controllers/ItemController.php
namespace App\Http\Controllers;

use App\Models\Item;
```

```
use Illuminate\Http\Request;
use Yajra\DataTables\Facades\DataTables;

class ItemController extends Controller
{
    public function index()
    {
        $categories = Item::select('category')->distinct()->pluck('category
');
        return view('items.index', compact('categories'));
    }

    public function getData(Request $request)
    {
        if ($request->ajax()) {
            $data = Item::query();

            // Category Filter
            if ($request->category && $request->category != 'all') {
                $data->where('category', $request->category);
            }

            // Price Range Filter
            if ($request->min_price) {
                $data->where('price', '>=', $request->min_price);
            }
            if ($request->max_price) {
                $data->where('price', '<=', $request->max_price);
            }

            return DataTables::of($data)
                ->addIndexColumn()
                ->addColumn('image', function($row){
                    return $row->image ? '<img src="/storage/'.$row->image.
'" width="50">' : 'No Image';
                })
                ->addColumn('price', function($row){
                    return '$'.number_format($row->price, 2);
                })
                ->filter(function ($query) use ($request) {
                    // Global Search
                    if ($request->has('search') && !empty($request->search[
'value'])) {
                        $search = $request->search['value'];
                        $query->where(function($q) use ($search) {
                            $q->where('name', 'like', "%{$search}%")
```

```
                            ->orWhere('category', 'like', "%{$search}%")
                            ->orWhere('price', 'like', "%{$search}%");
                    });
                }
            })
            ->rawColumns(['image'])
            ->make(true);
    }
}
}
```

✓ Blade View ဖန်တီးခြင်း

Code:

```
<!-- resources/views/items/index.blade.php -->
@extends('layouts.app')

@section('content')
<div class="container">
    <h1 class="mb-4">Item Management with DataTable</h1>

    <div class="card">
        <div class="card-header">
            <div class="row">
                <div class="col-md-3">
                    <label>Category Filter:</label>
                    <select class="form-control" id="categoryFilter">
                        <option value="all">All Categories</option>
                        @foreach($categories as $category)
                            <option value="{{ $category }}">{{ $category }}
</option>
                        @endforeach
                    </select>
                </div>
                <div class="col-md-3">
                    <label>Min Price:</label>
                    <input type="number" class="form-control" id="minPrice"
placeholder="Min price">
                </div>
                <div class="col-md-3">
                    <label>Max Price:</label>
                    <input type="number" class="form-control" id="maxPrice"
placeholder="Max price">
```

```
                </div>
                <div class="col-md-3">
                    <button class="btn btn-primary mt-4" id="filterBtn">Fil
ter</button>
                    <button class="btn btn-secondary mt-4" id="resetFilter"
>Reset</button>
                </div>
            </div>
        </div>
        <div class="card-body">
            <table class="table table-bordered" id="itemsTable" width="100%
">
                <thead>
                    <tr>
                        <th>No</th>
                        <th>Category</th>
                        <th>Name</th>
                        <th>Price</th>
                        <th>Image</th>
                    </tr>
                </thead>
                <tbody>
                    <!-- Data will be loaded via AJAX -->
                </tbody>
            </table>
        </div>
    </div>
</div>
@endsection

@section('scripts')
<!-- DataTables CSS and JS -->
<link rel="stylesheet" href="https://cdn.datatables.net/1.10.25/css/dataTab
les.bootstrap4.min.css">
<link rel="stylesheet" href="https://cdn.datatables.net/buttons/1.7.1/css/b
uttons.bootstrap4.min.css">
<script src="https://cdn.datatables.net/1.10.25/js/jquery.dataTables.min.js
"></script>
<script src="https://cdn.datatables.net/1.10.25/js/dataTables.bootstrap4.mi
n.js"></script>
<script src="https://cdn.datatables.net/buttons/1.7.1/js/dataTables.buttons
.min.js"></script>
<script src="https://cdn.datatables.net/buttons/1.7.1/js/buttons.bootstrap4
.min.js"></script>
<script src="https://cdn.datatables.net/buttons/1.7.1/js/buttons.html5.min.
js"></script>
```

```
<script src="https://cdn.datatables.net/buttons/1.7.1/js/buttons.print.min.
js"></script>

<script>
$(document).ready(function() {
    // Initialize DataTable
    var table = $('#itemsTable').DataTable({
        processing: true,
        serverSide: true,
        ajax: {
            url: "{{ route('items.data') }}",
            data: function (d) {
                d.category = $('#categoryFilter').val();
                d.min_price = $('#minPrice').val();
                d.max_price = $('#maxPrice').val();
            }
        },
        columns: [
            {data: 'DT_RowIndex', name: 'DT_RowIndex', orderable: false},
            {data: 'category', name: 'category'},
            {data: 'name', name: 'name'},
            {data: 'price', name: 'price', orderable: true},
            {data: 'image', name: 'image', orderable: false, searchable: fa
lse},
        ],
        dom: 'Bfrtip',
        buttons: [
            'copy', 'csv', 'excel', 'pdf', 'print'
        ],
        order: [[1, 'asc']] // Default sort by category ascending
    });

    // Apply Filter
    $('#filterBtn').click(function(){
        table.draw();
    });

    // Reset Filter
    $('#resetFilter').click(function(){
        $('#categoryFilter').val('all');
        $('#minPrice').val('');
        $('#maxPrice').val('');
        table.draw();
    });

    // Category Filter Change
```

```
    $('#categoryFilter').change(function(){
        table.draw();
    });
});
</script>
@endsection
```

✓ DataTable Package Config ပြုလုပ်ပုံ အဆင့်ဆင့်

1. Yajra DataTables Package Installation:

   Command:

```
composer require yajra/laravel-datatables-oracle
```

2. Configuration Setup:

   Code:

```
// config/app.php
'providers' => [
    // ...
    Yajra\DataTables\DataTablesServiceProvider::class,
];

'aliases' => [
    // ...
    'DataTables' => Yajra\DataTables\Facades\DataTables::class,
];
```

3. Publish Configuration:

   Command:

```
php artisan vendor:publish --provider="Yajra\DataTables\DataTablesSer
viceProvider"
```

4. Database Seeding:

   Command:

```
php artisan migrate --seed
```

✓ Application Features

1. Server-side Processing:
    - Data များကို server-side တွင် process လုပ်ပြီး pagination, searching, sorting တို့ကို အလိုအလျောက် handle လုပ်ပေးပါတယ်
2. Advanced Filtering:
    - Category dropdown filter
    - Price range filter (min and max)
    - Global search (DataTable built-in)
3. Sorting:
    - Column-based sorting (click on column headers)
    - Default sorting by category
4. Export Features:
    - Export to CSV, Excel, PDF
    - Print functionality
5. Responsive Design:
    - Bootstrap နဲ့ DataTable ကိုသုံးထားတာကြောင့် mobile နဲ့ tablet တွေမှာလည်း ကောင်းမွန်စွာ အလုပ်လုပ်ပါတယ်
6. Image Handling:
    - Image များကို table တွင် preview ကြည့်နိုင်ပါတယ်

✓ အသုံးပြုပုံ အဆင့်ဆင့်

1. Database migration နဲ့ model ကို create လုပ်ပါ

   Command:

```
php artisan make:model Item -m
```

2. Migration file ကို edit လုပ်ပြီး up() method ကို update လုပ်ပါ
3. Model ကို edit လုပ်ပြီး $fillable property ကို set လုပ်ပါ
4. Seeder ဖန်တီးပါ

   Command:

```
php artisan make:seeder ItemSeeder
```

5. Controller create လုပ်ပါ

Command:

```
php artisan make:controller ItemController
```

6. Route တွေကို `web.php` မှာ define လုပ်ပါ
7. View file ကို create လုပ်ပါ (`resources/views/items/index.blade.php`)
8. Storage link create လုပ်ပါ

Command:

```
php artisan storage:link
```

9. Application ကို run ပါ

Command:

```
php artisan serve
```

SHOPPING CART

Shopping Cart

Shopping Cart ဆိုတာ အွန်လိုင်းစတိုးတစ်ခုမှာ ဝယ်ယူလိုသော ပစ္စည်းများကို ယာယီသိမ်းဆည်းပေးသည့် System တစ်ခုဖြစ်ပါတယ်။ Laravel တွင် Shopping Cart ကို အောက်ပါ အဆင့်များ ဖြင့် အကောင်အထည်ဖော်နိုင်ပါတယ်။

✓ အခြေခံ Database Structure

Products Table

Code:

```
Schema::create('products', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->decimal('price', 8, 2);
    $table->text('description');
    $table->string('image')->nullable();
    $table->timestamps();
});
```

✓ Cart Table

Code:

```
Schema::create('carts', function (Blueprint $table) {
    $table->id();
    $table->unsignedBigInteger('user_id');
    $table->unsignedBigInteger('product_id');
    $table->integer('quantity');
    $table->timestamps();

    $table->foreign('user_id')->references('id')->on('users');
    $table->foreign('product_id')->references('id')->on('products');
});
```

✓ Model Relationships

■ Product Model

Code:

```
class Product extends Model
{
    protected $fillable = ['name', 'price', 'description', 'image'];
}
```

■ Cart Model

Code:

```
class Cart extends Model
{
    protected $fillable = ['user_id', 'product_id', 'quantity'];

    public function product()
    {
        return $this->belongsTo(Product::class);
    }

    public function user()
    {
        return $this->belongsTo(User::class);
    }
}
```

✓ Controller Logic

■ CartController

Code:

```
class CartController extends Controller
{
    public function index()
    {
        $cartItems = auth()->user()->cartItems;
        return view('cart.index', compact('cartItems'));
    }
```

```
    public function addToCart(Request $request, $productId)
    {
        $product = Product::findOrFail($productId);
        $cart = Cart::firstOrNew([
            'user_id' => auth()->id(),
            'product_id' => $product->id
        ]);

        $cart->quantity = $request->quantity ?? 1;
        $cart->save();

        return redirect()->back()->with('success', 'Product added to cart!'
);
    }

    public function updateCart(Request $request, $cartId)
    {
        $cart = Cart::findOrFail($cartId);
        $cart->update(['quantity' => $request->quantity]);
        return redirect()->back();
    }

    public function removeFromCart($cartId)
    {
        Cart::destroy($cartId);
        return redirect()->back();
    }
}
```

✓ Route များ သတ်မှတ်ခြင်း

Code:

```
Route::middleware(['auth'])->group(function () {
    Route::get('/cart', [CartController::class, 'index'])->name('cart.index
');
    Route::post('/cart/add/{product}', [CartController::class, 'addToCart']
)->name('cart.add');
    Route::put('/cart/update/{cart}', [CartController::class, 'updateCart']
)->name('cart.update');
    Route::delete('/cart/remove/{cart}', [CartController::class, 'removeFro
mCart'])->name('cart.remove');
});
```

✓ Blade Views

■ Product Listing

Code:

```
@foreach($products as $product)
<div class="product">
    <img src="{{ asset('storage/'.$product->image) }}" width="100">
    <h3>{{ $product->name }}</h3>
    <p>${{ $product->price }}</p>
    <form action="{{ route('cart.add', $product->id) }}" method="POST">
        @csrf
        <input type="number" name="quantity" value="1" min="1">
        <button type="submit">Add to Cart</button>
    </form>
</div>
@endforeach
```

■ Cart Page

Code:

```
<table class="cart-table">
    <thead>
        <tr>
            <th>Product</th>
            <th>Price</th>
            <th>Quantity</th>
            <th>Total</th>
            <th>Action</th>
        </tr>
    </thead>
    <tbody>
        @foreach($cartItems as $item)
        <tr>
            <td>{{ $item->product->name }}</td>
            <td>${{ $item->product->price }}</td>
            <td>
                <form action="{{ route('cart.update', $item->id) }}" method
="POST">
                    @csrf @method('PUT')
                    <input type="number" name="quantity" value="{{ $item->q
uantity }}" min="1">
                </form>
            </td>
```

```
            <td>${{ $item->product->price * $item->quantity }}</td>
            <td>
                <form action="{{ route('cart.remove', $item->id) }}" method
="POST">
                    @csrf @method('DELETE')
                    <button type="submit">Remove</button>
                </form>
            </td>
        </tr>
        @endforeach
    </tbody>
    <tfoot>
        <tr>
            <td colspan="3">Grand Total</td>
            <td>${{ $grandTotal }}</td>
            <td><a href="{{ route('checkout') }}">Checkout</a></td>
        </tr>
    </tfoot>
</table>
```

✓ အဆင့်မြင့် shopping cart feature တွေကို အောက်ပါအတိုင်း အဆင့်ဆင့် လေ့လာသင့်ပါတယ်။

1. Session-based Cart - Database အစား session ကိုသုံးပြီး cart လုပ်နိုင်ပါတယ်
2. Cart Package - darryldecode/cart ကဲ့သို့ package များသုံးနိုင်ပါတယ်
3. AJAX Cart - Page refresh မလုပ်စေဘဲ cart operations လုပ်နိုင်ပါတယ်
4. Multiple Currency - ငွေကြေးအမျိုးမျိုးကို ပံ့ပိုးနိုင်ပါတယ်
5. Discount System - Coupon code များနဲ့ လျှော့ဈေးစနစ်

✓ လိုအပ်သော Packages များ

1. Laravel UI (Authentication အတွက်)

   Command:

   ```
   composer require laravel/ui
   php artisan ui bootstrap --auth
   npm install && npm run dev
   ```

2. Darryldecode Cart (Optional)

   Command:

```
composer require darryldecode/cart
```

✓ Shopping cart ဆောက်တဲ့အခါ ထည့်သွင်းစဉ်းစားရမယ့် အချက်များ

1. Authentication - Cart operations အားလုံးကို auth middleware နဲ့ protect လုပ်ပါ
2. Validation - Quantity တွေကို server-side validation လုပ်ပါ
3. CSRF Protection - Form တွေမှာ CSRF token မဖြစ်မနေထည့်ပါ
4. Authorization - User တစ်ယောက်က အခြားသူ့ cart item တွေကို မဝင်ရောက်နိုင်စေရန်

Laravel Shopping Cart application ကို ဤအခြေခံအုတ်မြစ်များဖြင့် စတင်တည် ဆောက်နိုင်ပါတယ်။ Project ၏ လိုအပ်ချက်အရ database design နှင့် features များကို ထပ်မံချဲ့ထွင်နိုင်ပါတယ်။

လေ့ကျင့်ရန်-

Laravel Shopping Cart Application - အပြည့်အစုံ

✓ Database Structure

```
database/
  migrations/
    2014_10_12_000000_create_users_table.php
    2014_10_12_100000_create_items_table.php
    2014_10_12_200000_create_orders_table.php
    2014_10_12_300000_create_order_items_table.php
```

✓ Model Structure

```
app/Models/
  User.php
  Item.php
```

```
  Order.php
  OrderItem.php
```

✓ Controller Structure

```
app/Http/Controllers/
  UserController.php
  OrderController.php
  CartController.php
```

✓ View Structure

```
resources/views/
  products/
    index.blade.php    # Product List Page
    show.blade.php     # Product Detail Page
  cart/
    index.blade.php    # Cart List Page
  checkout/
    index.blade.php    # Checkout Page
    success.blade.php  # Checkout Success Page
```

☑ Database Migrations

✓ Users Table

Code:

```
Schema::create('users', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->string('email')->unique();
    $table->string('password');
    $table->string('image')->nullable();
    $table->rememberToken();
    $table->timestamps();
});
```

✓ Items Table

Code:

```
Schema::create('items', function (Blueprint $table) {
    $table->id();
```

```
    $table->string('category');
    $table->string('name');
    $table->decimal('price', 8, 2);
    $table->string('image')->nullable();
    $table->timestamps();
});
```

✓ Orders Table

Code:

```
Schema::create('orders', function (Blueprint $table) {
    $table->id();
    $table->string('order_no')->unique();
    $table->dateTime('order_date');
    $table->decimal('grand_total', 10, 2);
    $table->enum('order_status', ['pending', 'processing', 'completed', 'ca
ncelled'])->default('pending');
    $table->foreignId('user_id')->constrained();
    $table->timestamps();
});
```

✓ Order Items Table

Code:

```
Schema::create('order_items', function (Blueprint $table) {
    $table->id();
    $table->foreignId('order_id')->constrained();
    $table->foreignId('item_id')->constrained();
    $table->integer('quantity');
    $table->decimal('price', 8, 2);
    $table->timestamps();
});
```

☑ Model Relationships

✓ User Model

Code:

```
class User extends Authenticatable
{
    public function orders()
    {
        return $this->hasMany(Order::class);
```

```
    }
}
```

✓ Item Model

Code:

```
class Item extends Model
{
    public function orderItems()
    {
        return $this->hasMany(OrderItem::class);
    }
}
```

✓ Order Model

Code:

```
class Order extends Model
{
    public function user()
    {
        return $this->belongsTo(User::class);
    }

    public function orderItems()
    {
        return $this->hasMany(OrderItem::class);
    }
}
```

✓ OrderItem Model

Code:

```
class OrderItem extends Model
{
    public function order()
    {
        return $this->belongsTo(Order::class);
    }

    public function item()
    {
```

```
        return $this->belongsTo(Item::class);
    }
}
```

☑ Controller Logic with AJAX

✓ CartController.php (AJAX Version)

Code:

```
class CartController extends Controller
{
    public function addToCart(Request $request)
    {
        $item = Item::findOrFail($request->item_id);

        $cart = session()->get('cart', []);

        if(isset($cart[$item->id])) {
            $cart[$item->id]['quantity']++;
        } else {
            $cart[$item->id] = [
                "name" => $item->name,
                "quantity" => 1,
                "price" => $item->price,
                "image" => $item->image
            ];
        }

        session()->put('cart', $cart);

        return response()->json([
            'success' => true,
            'cart_count' => count($cart),
            'message' => 'Product added to cart successfully!'
        ]);
    }

    public function updateCart(Request $request)
    {
        if($request->item_id && $request->quantity){
            $cart = session()->get('cart');
            $cart[$request->item_id]["quantity"] = $request->quantity;
            session()->put('cart', $cart);
```

```
            return response()->json([
                'success' => true,
                'message' => 'Cart updated successfully'
            ]);
        }
    }

    public function removeFromCart(Request $request)
    {
        if($request->item_id) {
            $cart = session()->get('cart');
            if(isset($cart[$request->item_id])) {
                unset($cart[$request->item_id]);
                session()->put('cart', $cart);
            }

            return response()->json([
                'success' => true,
                'cart_count' => count($cart),
                'message' => 'Product removed successfully'
            ]);
        }
    }
}
```

✓ OrderController.php

Code:

```
class OrderController extends Controller
{
    public function checkout(Request $request)
    {
        $cart = session()->get('cart');
        $grandTotal = 0;

        foreach($cart as $id => $details){
            $grandTotal += $details['price'] * $details['quantity'];
        }

        $order = Order::create([
            'order_no' => 'ORD-'.strtoupper(uniqid()),
            'order_date' => now(),
            'grand_total' => $grandTotal,
            'user_id' => auth()->id()
        ]);
```

```
        foreach($cart as $id => $details){
            OrderItem::create([
                'order_id' => $order->id,
                'item_id' => $id,
                'quantity' => $details['quantity'],
                'price' => $details['price']
            ]);
        }

        session()->forget('cart');

        return response()->json([
            'success' => true,
            'order_id' => $order->id,
            'redirect_url' => route('checkout.success', $order->id)
        ]);
    }
}
```

☑ Routes (web.php)

Code:

```
Route::middleware(['auth'])->group(function () {
    // Product Routes
    Route::get('/products', [ProductController::class, 'index'])->name('pro
ducts.index');
    Route::get('/products/{id}', [ProductController::class, 'show'])->name(
'products.show');

    // Cart Routes (AJAX)
    Route::post('/cart/add', [CartController::class, 'addToCart'])->name('c
art.add');
    Route::put('/cart/update', [CartController::class, 'updateCart'])->name
('cart.update');
    Route::delete('/cart/remove', [CartController::class, 'removeFromCart']
)->name('cart.remove');
    Route::get('/cart', [CartController::class, 'index'])->name('cart.index
');

    // Checkout Routes
    Route::post('/checkout', [OrderController::class, 'checkout'])->name('c
heckout.process');
    Route::get('/checkout/success/{order}', [OrderController::class, 'succe
ss'])->name('checkout.success');
```

```
});
```

☑ Views with AJAX Integration

✓ Product List Page (products/index.blade.php)

Code:

```
@foreach($items as $item)
<div class="product-item">
    <img src="{{ asset('storage/'.$item->image) }}" width="100">
    <h3>{{ $item->name }}</h3>
    <p>${{ number_format($item->price, 2) }}</p>
    <button class="btn btn-primary add-to-cart" data-item-id="{{ $item->id
}}">Add to Cart</button>
    <a href="{{ route('products.show', $item->id) }}" class="btn btn-info">
View Details</a>
</div>
@endforeach
```

```
<script>
$(document).ready(function(){
    $('.add-to-cart').click(function(){
        let itemId = $(this).data('item-id');

        $.ajax({
            url: "{{ route('cart.add') }}",
            method: "POST",
            data: {
                item_id: itemId,
                _token: '{{ csrf_token() }}'
            },
            success: function(response){
                $('#cart-count').text(response.cart_count);
                alert(response.message);
            },
            error: function(xhr){
                alert('Error: ' + xhr.responseJSON.message);
            }
        });
    });
});
</script>
```

✓ Cart Page (cart/index.blade.php)

Code:

```
<table class="table cart-table">
    <tbody>
        @foreach($cart as $id => $details)
        <tr data-item-id="{{ $id }}">
            <td><img src="{{ asset('storage/'.$details['image']) }}" width=
"50"></td>
            <td>{{ $details['name'] }}</td>
            <td>${{ number_format($details['price'], 2) }}</td>
            <td>
                <input type="number" class="quantity" value="{{ $details['q
uantity'] }}" min="1">
            </td>
            <td>${{ number_format($details['price'] * $details['quantity'],
2) }}</td>
            <td>
                <button class="btn btn-danger remove-from-cart">Remove</but
ton>
            </td>
        </tr>
        @endforeach
    </tbody>
</table>

<button id="checkout-btn" class="btn btn-success">Proceed to Checkout</butt
on>

<script>
$(document).ready(function(){
    // Update quantity
    $('.quantity').change(function(){
        let itemId = $(this).closest('tr').data('item-id');
        let quantity = $(this).val();

        $.ajax({
            url: "{{ route('cart.update') }}",
            method: "PUT",
            data: {
                item_id: itemId,
                quantity: quantity,
                _token: '{{ csrf_token() }}'
            },
            success: function(response){
```

```
                location.reload();
            }
        });
    });

    // Remove item
    $('.remove-from-cart').click(function(){
        let itemId = $(this).closest('tr').data('item-id');

        $.ajax({
            url: "{{ route('cart.remove') }}",
            method: "DELETE",
            data: {
                item_id: itemId,
                _token: '{{ csrf_token() }}'
            },
            success: function(response){
                location.reload();
            }
        });
    });

    // Checkout
    $('#checkout-btn').click(function(){
        $.ajax({
            url: "{{ route('checkout.process') }}",
            method: "POST",
            data: {
                _token: '{{ csrf_token() }}'
            },
            success: function(response){
                window.location.href = response.redirect_url;
            }
        });
    });
});
</script>
```

☑ Project Testing

✓ Unit Tests

Code:

```
class CartTest extends TestCase
```

```
{
    public function test_add_to_cart()
    {
        $user = User::factory()->create();
        $item = Item::factory()->create();

        $response = $this->actingAs($user)
                         ->postJson(route('cart.add'), ['item_id' => $item-
>id]);

        $response->assertStatus(200)
                 ->assertJson(['success' => true]);
    }

    public function test_checkout_process()
    {
        $user = User::factory()->create();
        $item = Item::factory()->create();

        // Add item to cart first
        $this->actingAs($user)
             ->postJson(route('cart.add'), ['item_id' => $item->id]);

        $response = $this->actingAs($user)
                         ->postJson(route('checkout.process'));

        $response->assertStatus(200)
                 ->assertJson(['success' => true]);

        $this->assertDatabaseHas('orders', ['user_id' => $user->id]);
    }
}
```

✓ Feature Tests

Code:

```
class CheckoutTest extends TestCase
{
    public function test_guest_cannot_access_checkout()
    {
        $response = $this->get(route('cart.index'));
        $response->assertRedirect('/login');
    }

    public function test_user_can_view_cart()
```

```
    {
        $user = User::factory()->create();

        $response = $this->actingAs($user)
                         ->get(route('cart.index'));

        $response->assertStatus(200);
    }
}
```

✓ အဆင့်မြင့် Features များ

1. Payment Gateway Integration - Stripe, PayPal တို့နဲ့ချိတ်ဆက်ခြင်း
2. Coupon/Discount System - လျှော့ဈေးကုဒ်များထည့်သွင်းခြင်း
3. Shipping Calculation - ပို့ဆောင်ခတွက်ချက်ခြင်း
4. Order Tracking - အမှာစာအခြေအနေကိုခြေရာခံခြင်း
5. Inventory Management - ပစ္စည်းလက်ကျန်စီမံခန့်ခွဲခြင်း

✓ Project Setup Steps

1. Database migration နှင့် seeding

   Command:

   ```
   php artisan migrate --seed
   ```

2. Storage link create

   Command:

   ```
   php artisan storage:link
   ```

3. Test run

   Command:

   ```
   php artisan test
   ```

4. Development server start

Command:

```
php artisan serve
```

ဤ application တွင် Laravel ၏ session-based cart system ကိုအသုံးပြုထားပြီး AJAX technology ဖြင့် page refresh မလုပ်စေဘဲ ချောမွေ့သော user experience ကိုဖန်တီးပေးထားပါတယ်။ Project ကို လိုအပ်သလို ချဲ့ထွင်အသုံးပြုနိုင်ပါတယ်။

AUTHENTICATION

Authentication

Laravel Authentication ဆိုတာ ဝက်ဘ်အပလီကေးရှင်းတစ်ခုမှာ အသုံးပြုသူများ (users) အား အတည်ပြုခြင်းနှင့် ခွင့်ပြုချက်ပေးခြင်း စနစ် ဖြစ်ပါတယ်။ Laravel မှာ authentication ကို အလွယ်တကူ အကောင်အထည်ဖော်နိုင်ပါတယ်။

✓ Authentication အတွက် အခြေခံ Setup

⇨ Laravel UI Package တပ်ဆင်ခြင်း

Command:

```
composer require laravel/ui
php artisan ui bootstrap --auth
npm install && npm run dev
```

✓ Database Migration

Code:

```
Schema::create('users', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->string('email')->unique();
    $table->timestamp('email_verified_at')->nullable();
    $table->string('password');
    $table->rememberToken();
    $table->timestamps();
});
```

✓ Authentication လုပ်ဆောင်ပုံ အဆင့်ဆင့်

⇨ User Registration

Code:

```
// routes/web.php
Route::get('/register', [RegisteredUserController::class, 'create'])
```

```
                ->middleware('guest')
                ->name('register');

Route::post('/register', [RegisteredUserController::class, 'store'])
                ->middleware('guest');
```

✓ User Login

Code:

```
Route::get('/login', [AuthenticatedSessionController::class, 'create'])
                ->middleware('guest')
                ->name('login');

Route::post('/login', [AuthenticatedSessionController::class, 'store'])
                ->middleware('guest');
```

✓ User Logout

Code:

```
Route::post('/logout', [AuthenticatedSessionController::class, 'destroy'])
                ->middleware('auth')
                ->name('logout');
```

✓ Authentication Controllers

⇨ Register Controller

Code:

```
protected function create(array $data)
{
    return User::create([
        'name' => $data['name'],
        'email' => $data['email'],
        'password' => Hash::make($data['password']),
    ]);
}
```

⇨ Login Controller

Code:

```
public function store(Request $request)
{
    $credentials = $request->validate([
```

```
        'email' => ['required', 'email'],
        'password' => ['required'],
    ]);

    if (Auth::attempt($credentials)) {
        $request->session()->regenerate();
        return redirect()->intended('dashboard');
    }

    return back()->withErrors([
        'email' => 'The provided credentials do not match our records.',
    ]);
}
```

✓ Middleware အသုံးပြုခြင်း

⇨ Authenticated Users အတွက်

Code:

```
Route::get('/profile', function () {
    // Only authenticated users may access this route...
})->middleware('auth');
```

⇨ Guest Users အတွက်

Code:

```
Route::get('/login', function () {
    // Only guests may access this route...
})->middleware('guest');
```

✓ Authentication နှင့် ဆက်စပ်သော Features များ

⇨ Password Reset

Command:

```
php artisan vendor:publish --tag=laravel-mail
```

⇨ Email Verification

Code:

```
class User extends Authenticatable implements MustVerifyEmail
{
    use Notifiable;
}
```

⇨ Remember Me Functionality

Code:

```
if (Auth::attempt(['email' => $email, 'password' => $password], $remember))
{
    // The user is being remembered...
}
```

✓ Authentication အတွက် Security Best Practices

1. Password Hashing - Laravel မှာ default အားဖြင့် bcrypt ကို အသုံးပြုထားပါတယ်
2. CSRF Protection - Form တိုင်းမှာ CSRF token ပါရှိပါတယ်
3. Rate Limiting - Login attempts ကို throttle လုပ်ထားပါတယ်
4. Secure Cookies - Session cookies များကို secure ဖြစ်အောင် configure လုပ်ထားပါတယ်

✓ Custom Authentication လုပ်ဆောင်နည်း

⇨ Custom Guard ဖန်တီးခြင်း

Config:

```
// config/auth.php
```

```
'guards' => [
    'admin' => [
        'driver' => 'session',
        'provider' => 'admins',
    ],
],
```

⇨ Custom User Provider

Config:

```
'providers' => [
    'admins' => [
        'driver' => 'eloquent',
        'model' => App\Models\Admin::class,
    ],
],
```

✓ Socialite Package အသုံးပြုခြင်း (Social Login)

⇨ Installation

Command:

```
composer require laravel/socialite
```

⇨ Configuration

Config:

```
// config/services.php
'github' => [
    'client_id' => env('GITHUB_CLIENT_ID'),
    'client_secret' => env('GITHUB_CLIENT_SECRET'),
    'redirect' => 'http://your-callback-url',
],
```

✓ Testing Authentication

⇨ Feature Test Example

Code:

```
public function test_login_screen_can_be_rendered()
{
    $response = $this->get('/login');
    $response->assertStatus(200);
}

public function test_users_can_authenticate()
{
    $user = User::factory()->create();

    $response = $this->post('/login', [
        'email' => $user->email,
        'password' => 'password',
    ]);

    $this->assertAuthenticated();
    $response->assertRedirect('/dashboard');
}
```

Laravel Authentication စနစ်သည် လုံခြုံစိတ်ချရပြီး စနစ်တကျဒီဇိုင်းဆွဲထားသော အတည်ပြုခြင်းစနစ်ဖြစ်ပါတယ်။ Project ၏ လိုအပ်ချက်အရ လွယ်ကူစွာ customize လုပ်နိုင်ပါတယ်။

☑ Google Login

✓ Laravel Socialite Package ထည့်သွင်းရပါမယ်။

Composer ဖြင့် Socialite တပ်ဆင်ပါ

Command:

```
composer require laravel/socialite
```

✓ Google Developer Console တွင် Project Setup

1. Google Developer Console သို့ သွားပါ
2. New Project ဖန်တီးပါ
3. "OAuth consent screen" configure လုပ်ပါ
4. "Credentials" တွင် "Create Credentials" > "OAuth client ID" ရွေးပါ
5. Web application အမျိုးအစားရွေးပါ
6. Authorized redirect URIs
   ထဲတွင် http://yourdomain.com/auth/google/`callback` ထည့်ပါ

✓ Laravel Project တွင် Configuration

.env File တွင် Google Credentials များထည့်ပါ

Code:

```
GOOGLE_CLIENT_ID=your_client_id
GOOGLE_CLIENT_SECRET=your_client_secret
GOOGLE_REDIRECT_URI=http://yourdomain.com/auth/google/callback
```

config/services.php တွင် Configuration ဖြည့်ပါ

Code:

```
'google' => [
    'client_id' => env('GOOGLE_CLIENT_ID'),
    'client_secret' => env('GOOGLE_CLIENT_SECRET'),
    'redirect' => env('GOOGLE_REDIRECT_URI'),
],
```

✓ Database Migration ပြင်ဆင်ခြင်း

users table ကို update လုပ်ပါ

Command:

```
php artisan make:migration add_google_id_to_users_table --table=users
```

Code:

```
public function up()
{
    Schema::table('users', function (Blueprint $table) {
        $table->string('google_id')->nullable();
        $table->string('avatar')->nullable();
    });
}
```

Command:

```
php artisan migrate
```

✓ Controller ဖန်တီးခြင်း

GoogleAuthController ဖန်တီးပါ

Command:

```
php artisan make:controller GoogleAuthController
```

Code:

```
namespace App\Http\Controllers;

use App\Models\User;
use Illuminate\Support\Facades\Auth;
```

```
use Laravel\Socialite\Facades\Socialite;

class GoogleAuthController extends Controller
{
    public function redirect()
    {
        return Socialite::driver('google')->redirect();
    }

    public function callback()
    {
        try {
            $googleUser = Socialite::driver('google')->user();

            $user = User::where('google_id', $googleUser->getId())->first()
;


            if(!$user) {
                $user = User::updateOrCreate([
                    'email' => $googleUser->getEmail()
                ], [
                    'name' => $googleUser->getName(),
                    'google_id' => $googleUser->getId(),
                    'avatar' => $googleUser->getAvatar(),
                    'password' => bcrypt(uniqid()) // Random password
                ]);
            }

            Auth::login($user);

            return redirect('/dashboard');

        } catch (\Exception $e) {
            return redirect('/login')->withErrors('Google login failed: '.$
e->getMessage());
        }
    }
}
```

✓ Routes သတ်မှတ်ခြင်း

Code:

```
Route::get('/auth/google', [GoogleAuthController::class, 'redirect'])->name
('google.login');
```

```
Route::get('/auth/google/callback', [GoogleAuthController::class, 'callback
']);
```

✓ View တွင် Login Button ထည့်သွင်းခြင်း

Code:

```
<a href="{{ route('google.login') }}" class="btn btn-google">
    <i class="fab fa-google"></i> Login with Google
</a>
```

✓ အဆင့်မြင့် Configuration များ

Scopes ထည့်ခြင်း

Code:

```
return Socialite::driver('google')
    ->scopes(['profile', 'email'])
    ->redirect();
```

Offline Access (Refresh Token)

Code:

```
return Socialite::driver('google')
    ->with(['access_type' => 'offline', 'prompt' => 'consent select_account
'])
    ->redirect();
```

✓ Security Considerations

1. State Parameter - CSRF protection အတွက်
2. Email Verification - Google မှ verified emails များကိုသုံးပါ
3. Token Validation - ID token များကို verify လုပ်ပါ

✓ Testing Google Login

Test Case Example

Code:

```
public function test_google_redirect_works()
{
    $response = $this->get('/auth/google');
    $response->assertRedirect();
}

public function test_google_callback_creates_user()
{
    $this->mock(Socialite::class, function ($mock) {
        $mock->shouldReceive('driver->user')
            ->andReturn((object) [
                'id' => '12345',
                'name' => 'Test User',
                'email' => 'test@example.com',
                'avatar' => 'avatar.jpg'
            ]);
    });

    $response = $this->get('/auth/google/callback');
    $response->assertRedirect('/dashboard');
    $this->assertDatabaseHas('users', ['email' => 'test@example.com']);
}
```

✓ Troubleshooting

1. Redirect URI mismatch - Google Console တွင် သတ်မှတ်ထားသော URI နှင့် တူညီစေရန်
2. Invalid permissions - Google API Console တွင် required scopes များ enable လုပ်ပါ
3. Token expiration - Refresh token များကို အသုံးပြုပါ

Laravel တွင် Google Login အား ဤနည်းလမ်းဖြင့် အလွယ်တကူ အကောင်အထည်ဖော်နိုင်ပါတယ်။ Socialite package သည် OAuth flow များကို အလိုအလျောက် handle လုပ်ပေးပြီး developer များအတွက် အလုပ်များကို များစွာ လျှော့ချပေးပါတယ်။

☑ Facebook Login

✓ Socialite Package ထည့်သွင်းရပါမယ်။

Command:

```
composer require laravel/socialite
```

✓ Facebook Developer Console တွင် App ဖန်တီးခြင်း

1. Facebook Developers သို့ ဝင်ပါ
2. "My Apps" > "Create App" ကိုနှိပ်ပါ
3. App Type အဖြစ် "Consumer" ကိုရွေးပါ
4. "Facebook Login" product ကိုထည့်ပါ
5. Valid OAuth Redirect URIs
   ထဲတွင် http://yourdomain.com/auth/facebook/callback ထည့်ပါ

✓ Laravel Project တွင် Configuration

.env File တွင် Facebook Credentials များထည့်ပါ

Code:

```
FACEBOOK_CLIENT_ID=your_app_id
FACEBOOK_CLIENT_SECRET=your_app_secret
FACEBOOK_REDIRECT_URI=http://yourdomain.com/auth/facebook/callback
```

config/services.php တွင် Configuration ဖြည့်ပါ

Code:

```
'facebook' => [
    'client_id' => env('FACEBOOK_CLIENT_ID'),
    'client_secret' => env('FACEBOOK_CLIENT_SECRET'),
    'redirect' => env('FACEBOOK_REDIRECT_URI'),
],
```

✓ Database Migration ပြင်ဆင်ခြင်း

users table ကို update လုပ်ပါ

Command:

```
php artisan make:migration add_facebook_fields_to_users_table --table=users
```

Code:

```
public function up()
{
    Schema::table('users', function (Blueprint $table) {
        $table->string('facebook_id')->nullable();
        $table->string('avatar')->nullable();
    });
}
```

Command:

```
php artisan migrate
```

✓ Controller ဖန်တီးခြင်း

FacebookAuthController ဖန်တီးပါ

Command:

```
php artisan make:controller FacebookAuthController
```

Code:

```
namespace App\Http\Controllers;

use App\Models\User;
use Illuminate\Support\Facades\Auth;
use Laravel\Socialite\Facades\Socialite;

class FacebookAuthController extends Controller
{
    public function redirect()
    {
        return Socialite::driver('facebook')->redirect();
    }

    public function callback()
    {
        try {
```

```
            $facebookUser = Socialite::driver('facebook')->user();

            $user = User::where('facebook_id', $facebookUser->getId())->fir
st();

            if(!$user) {
                $user = User::updateOrCreate([
                    'email' => $facebookUser->getEmail()
                ], [
                    'name' => $facebookUser->getName(),
                    'facebook_id' => $facebookUser->getId(),
                    'avatar' => $facebookUser->getAvatar(),
                    'password' => bcrypt(uniqid()) // Random password
                ]);
            }

            Auth::login($user);

            return redirect('/dashboard');

        } catch (\Exception $e) {
            return redirect('/login')->withErrors('Facebook login failed: '
.$e->getMessage());
        }
    }
}
```

✓ Routes သတ်မှတ်ခြင်း

Code:

```
Route::get('/auth/facebook', [FacebookAuthController::class, 'redirect'])->
name('facebook.login');
Route::get('/auth/facebook/callback', [FacebookAuthController::class, 'call
back']);
```

✓ View တွင် Login Button ထည့်သွင်းခြင်း

Code:

```
<a href="{{ route('facebook.login') }}" class="btn btn-facebook">
    <i class="fab fa-facebook"></i> Login with Facebook
</a>
```

✓ အဆင့်မြင့် Configuration များ

Scopes ထည့်ခြင်း

Code:

```
return Socialite::driver('facebook')
    ->scopes(['email', 'public_profile'])
    ->redirect();
```

✓ Fields များ Request လုပ်ခြင်း

Code:

```
return Socialite::driver('facebook')
    ->fields(['name', 'first_name', 'last_name', 'email', 'picture'])
    ->redirect();
```

✓ Security Considerations

1. App Secret Proof - Facebook API calls များတွင် ထည့်သွင်းပါ
2. Email Verification - Facebook မှ verified emails များကိုသာ အသုံးပြုပါ
3. Token Validation - Access tokens များကို verify လုပ်ပါ

✓ Testing Facebook Login

Test Case Example

Code:

```
public function test_facebook_redirect_works()
{
    $response = $this->get('/auth/facebook');
    $response->assertRedirect();
}

public function test_facebook_callback_creates_user()
{
    $this->mock(Socialite::class, function ($mock) {
        $mock->shouldReceive('driver->user')
            ->andReturn((object) [
                'id' => '12345',
                'name' => 'Test User',
                'email' => 'test@example.com',
```

```
                'avatar' => 'avatar.jpg'
            ]);
    });

    $response = $this->get('/auth/facebook/callback');
    $response->assertRedirect('/dashboard');
    $this->assertDatabaseHas('users', ['email' => 'test@example.com']);
}
```

✓ Troubleshooting

1. Redirect URI mismatch - Facebook App Settings တွင် သတ်မှတ်ထားသော URI နှင့် တူညီစေရန်
2. App Review Required - Facebook တွင် app review လုပ်ပါ
3. Email Permission - email permission ကို request လုပ်ပါ

Laravel တွင် Facebook Login အား ဤနည်းလမ်းဖြင့် အလွယ်တကူ အကောင်အထည် ဖော်နိုင်ပါတယ်။ Socialite package သည် OAuth flow များကို အလိုအလျောက် handle လုပ်ပေးပြီး developer များအတွက် အလုပ်များကို လျှော့ချပေးပါတယ်။

☑ Github Login

✓ Socialite Package ထည့်သွင်းရပါမယ်။

Command:

```
composer require laravel/socialite
```

✓ GitHub Developer Settings တွင် OAuth App ဖန်တီးခြင်း

1. GitHub Developer Settings သို့ ဝင်ပါ
2. "OAuth Apps" > "New OAuth App" ကိုနှိပ်ပါ
3. Application name, Homepage URL ထည့်ပါ
4. Authorization callback URL
   ထဲတွင် http://yourdomain.com/auth/github/callback ထည့်ပါ
5. "Register application" ကိုနှိပ်ပါ

✓ Laravel Project တွင် Configuration

.env File တွင် GitHub Credentials များထည့်ပါ

Code:

```
GITHUB_CLIENT_ID=your_github_client_id
GITHUB_CLIENT_SECRET=your_github_client_secret
GITHUB_REDIRECT_URI=http://yourdomain.com/auth/github/callback
```

✓ config/services.php တွင် Configuration ဖြည့်ပါ

Code:

```
'github' => [
    'client_id' => env('GITHUB_CLIENT_ID'),
    'client_secret' => env('GITHUB_CLIENT_SECRET'),
    'redirect' => env('GITHUB_REDIRECT_URI'),
],
```

✓ Database Migration ပြင်ဆင်ခြင်း

■ users table ကို update လုပ်ပါ

Command:

```
php artisan make:migration add_github_fields_to_users_table --table=users
```

Code:

```
public function up()
{
    Schema::table('users', function (Blueprint $table) {
        $table->string('github_id')->nullable();
        $table->string('avatar')->nullable();
    });
}
```

Command:

```
php artisan migrate
```

✓ Controller ဖန်တီးခြင်း

■ GitHubAuthController ဖန်တီးကြပါမယ်။

Command:

```
php artisan make:controller GitHubAuthController
```

Code:

```
namespace App\Http\Controllers;

use App\Models\User;
use Illuminate\Support\Facades\Auth;
use Laravel\Socialite\Facades\Socialite;

class GitHubAuthController extends Controller
{
    public function redirect()
    {
        return Socialite::driver('github')->redirect();
    }

    public function callback()
    {
        try {
            $githubUser = Socialite::driver('github')->user();

            $user = User::where('github_id', $githubUser->getId())->first()
;

            if(!$user) {
```

```
                $user = User::updateOrCreate([
                    'email' => $githubUser->getEmail()
                ], [
                    'name' => $githubUser->getName() ?? $githubUser->getNic
kname(),
                    'github_id' => $githubUser->getId(),
                    'avatar' => $githubUser->getAvatar(),
                    'password' => bcrypt(uniqid()) // Random password
                ]);
            }

            Auth::login($user);

            return redirect('/dashboard');

        } catch (\Exception $e) {
            return redirect('/login')->withErrors('GitHub login failed: '.$
e->getMessage());
        }
    }
}
```

✓ Routes တွေကို အောက်ပါအတိုင်းသတ်မှတ်ရပါမယ်။

Code:

```
Route::get('/auth/github', [GitHubAuthController::class, 'redirect'])->name
('github.login');
Route::get('/auth/github/callback', [GitHubAuthController::class, 'callback
']);
```

✓ View မှာ Login Button ကို အောက်ပါအတိုင်းထည့်သွင်းကြပါမယ်။

Code:

```
<a href="{{ route('github.login') }}" class="btn btn-github">
    <i class="fab fa-github"></i> Login with GitHub
</a>
```

✓ အဆင့်မြင့် Configuration များ

■ Scopes ထည့်ခြင်း

Code:

```
return Socialite::driver('github')
    ->scopes(['user:email', 'read:user'])
    ->redirect();
```

✓ User Details များ Request လုပ်ခြင်း

Code:

```
$user = Socialite::driver('github')->user();
$user->getNickname();  // GitHub username
$user->getAvatar();    // Profile image
$user->user['bio'];    // User bio
```

✓ Security Considerations

1. State Parameter - CSRF protection အတွက်
2. Email Verification - GitHub မှ primary email ကိုသာ အသုံးပြုပါ
3. Token Validation - Access tokens များကို verify လုပ်ပါ

✓ Testing GitHub Login

■ Test Case Example

Code:

```
public function test_github_redirect_works()
{
    $response = $this->get('/auth/github');
    $response->assertRedirect();
}
```

```
public function test_github_callback_creates_user()
{
    $this->mock(Socialite::class, function ($mock) {
        $mock->shouldReceive('driver->user')
            ->andReturn((object) [
                'id' => '12345',
                'nickname' => 'testuser',
```

```
                'name' => 'Test User',
                'email' => 'test@example.com',
                'avatar' => 'avatar.jpg'
            ]);
    });

    $response = $this->get('/auth/github/callback');
    $response->assertRedirect('/dashboard');
    $this->assertDatabaseHas('users', ['email' => 'test@example.com']);
}
```

✓ Troubleshooting

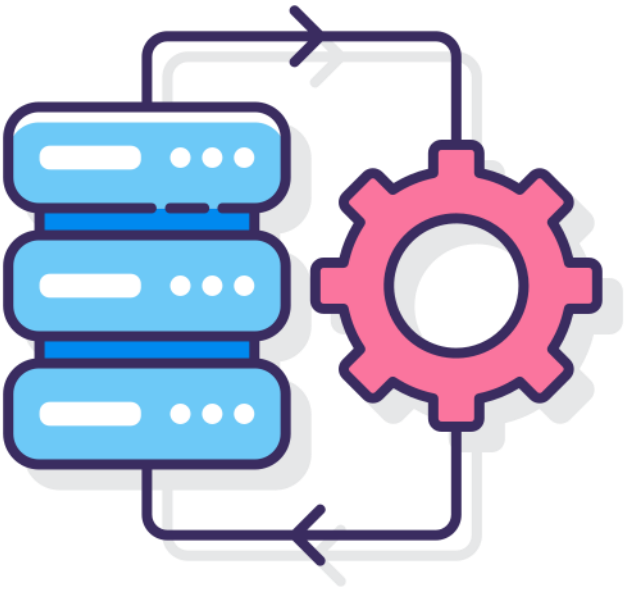
1. Redirect URI mismatch - GitHub OAuth App Settings တွင် သတ်မှတ်ထားသော URI နှင့် တူညီစေရန်
2. Email permission - user:email scope ထည့်သွင်းပါ
3. Rate limiting - GitHub API rate limits ကို သတိပြုပါ

Laravel တွင် GitHub Login အား ဤနည်းလမ်းဖြင့် အလွယ်တကူ အကောင်အထည်ဖော်နိုင်ပါတယ်။ Socialite package သည် OAuth flow များကို အလိုအလျောက် handle လုပ်ပေးပြီး developer များအတွက် အလုပ်များကို လျှော့ချပေးပါတယ်။

GitHub Login သည် developer များအတွက် အထူးသင့်တော်ပြီး၊ အောက်ပါအချက်များကို သတိပြုပါ:

1. GitHub သည် email address များကို private ထားနိုင်သောကြောင့် email ရရှိရန် user:email scope လိုအပ်ပါမယ်
2. GitHub username (nickname) ကို default name အဖြစ် အသုံးပြုသင့်ပါတယ်
3. Public profile information များကို အလွယ်တကူ ရယူနိုင်ပါတယ်

MIDDLEWARE



Middleware

Middleware ဆိုတာက Laravel application တစ်ခုမှာ HTTP request တွေ response တွေ စီးဆင်းတဲ့အခါ အလိုအလျောက် လုပ်ဆောင်ပေးမယ့် layer တစ်ခုဖြစ်ပါတယ်။ ဒါကို "ကြားခံအလွှာ" လို့လည်း ခေါ်နိုင်ပါတယ်။ Middleware ကို သုံးရတဲ့ အဓိကအကြောင်းရင်း တွေကတော့ Request မှာ authentication (အတည်ပြုခြင်း) လုပ်ဖို့၊ CSRF protection (ကြားခံတိုက်ခိုက်မှုကာကွယ်ခြင်း) လုပ်ပေးဖို့၊ Maintenance mode (ပြုပြင်ထိန်းသိမ်းမှုမုဒ်) မှာ ရှိနေလားဆိုတာ စစ်ဖို့၊ Request data တွေကို စစ်ထုတ်ဖို့ (sanitize/filter) နဲ့ Response တွေကို modify လုပ်ဖို့ ဖြစ်ပါတယ်။

Laravel မှာ middleware အမျိုးအစား ၃ မျိုးရှိပါတယ်:

1. Global Middleware - Request တိုင်းအတွက် အလုပ်လုပ်မယ်
2. Route Middleware - သတ်မှတ်ထားတဲ့ route တွေမှာပဲ အလုပ်လုပ်မယ်
3. Group Middleware - Middleware group တွေဖွဲ့ပြီး သုံးနိုင်တယ်

✓ Middleware ဖန်တီးနည်း

Middleware အသစ်ဖန်တီးဖို့ terminal မှာ အောက်ပါ command ကို ရိုက်ထည့်ပါ:

Code:

```
php artisan make:middleware CheckAge
```

ဒါဆိုရင် app/Http/Middleware folder အောက်မှာ CheckAge.php file အသစ်ဖန်တီးပေးပါလိမ့်မယ်။

Middleware Structure

ဥပမာ CheckAge middleware ရဲ့ structure က အောက်ပါအတိုင်းဖြစ်ပါတယ်:

Code:

```
<?php

namespace App\Http\Middleware;
```

```
use Closure;

class CheckAge
{
    public function handle($request, Closure $next)
    {
        if ($request->age <= 18) {
            return redirect('home');
        }

        return $next($request);
    }
}
```

✓ Middleware ကို Register လုပ်နည်း

Middleware အသစ်ဖန်တီးပြီးရင် app/Http/Kernel.php မှာ register လုပ်ဖို့လိုပါတယ်။

Code:

```
protected $routeMiddleware = [
    // ...
    'age' => \App\Http\Middleware\CheckAge::class,
];
```

✓ Middleware ကို Route တွေမှာ သုံးနည်း

⇨ Single Middleware

Code:

```
Route::get('admin/profile', function () {
    //
})->middleware('auth');
```

⇨ Multiple Middleware

Code:

```
Route::get('/', function () {
    //
})->middleware(['first', 'second']);
```

⇨ Middleware Group

Code:

```
Route::middleware(['web'])->group(function () {
    Route::get('/', function () {
        // Uses 'web' middleware group
    });
});
```

⇨ Middleware Parameters

Middleware ကို parameter တွေနဲ့လည်း သုံးနိုင်ပါတယ်။

Code:

```
Route::put('post/{id}', function ($id) {
    //
})->middleware('role:editor');
```

Middleware မှာ parameter ကို လက်ခံမယ့် ပုံစံ:

Code:

```
public function handle($request, Closure $next, $role)
{
    if (! $request->user()->hasRole($role)) {
        // Redirect...
    }

    return $next($request);
}
```

⇨ Terminable Middleware

တခါတလေ middleware က response ပြီးတဲ့အခါမှာ အလုပ်လုပ်ဖို့လိုတတ်ပါတယ်။
အဲ့ဒီအတွက် terminate method ကို သုံးနိုင်ပါတယ်။

Code:

```
public function terminate($request, $response)
{
    // Response ပြီးမှ အလုပ်လုပ်မယ့် code တွေ
}
```

✓ Middleware အသုံးပြုပုံ

■ Language Middleware

Code:

```
public function handle($request, Closure $next)
{
    if (session()->has('locale')) {
        app()->setLocale(session()->locale);
    }
    return $next($request);
}
```

■ Admin Check Middleware

Code:

```
public function handle($request, Closure $next)
{
    if (auth()->check() && auth()->user()->is_admin) {
        return $next($request);
    }
    return redirect('/home');
}
```

Laravel middleware တွေက application ရဲ့ request/response lifecycle မှာ အရေးကြီးတဲ့အခန်းကဏ္ဍက ပါဝင်ပါတယ်။ Middleware တွေကို ကောင်းကောင်းသုံးတတ်ရင် application logic တွေကို ပိုမိုသန့်ရှင်းစွာ ရေးသားနိုင်ပြီး ထိန်းသိမ်းရလွယ်ကူစေပါတယ်။

လေ့ကျင့်ရန် -

✓ Middleware ဖန်တီးခြင်း

ပထမဆုံး CheckAdmin middleware ကို ဖန်တီးပါမယ်။ Terminal မှာ အောက်ပါ command ကို ရိုက်ထည့်ပါ။

Code:

```
php artisan make:middleware CheckAdmin
```

✓ CheckAdmin Middleware ကို ပြင်ဆင်ခြင်း

■ app/Http/Middleware/CheckAdmin.php ဖိုင်ကို အောက်ပါအတိုင်း ပြင်ဆင်ပါမယ်။

Code:

```
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;

class CheckAdmin
{
    public function handle(Request $request, Closure $next)
    {
        // User ဝင်မဝင် စစ်ဆေးခြင်း
        if (!Auth::check()) {
            return redirect()->route('login');
        }
```

```
        // User type အလိုက် redirect လုပ်ခြင်း
        $user = Auth::user();

        if ($user->usertype === 'admin') {
            return redirect()->route('admin.dashboard');
        }

        if ($user->usertype === 'user') {
            return redirect()->route('user.dashboard');
        }

        return $next($request);
    }
}
```

✓ Middleware ကို Register လုပ်ခြင်း

■ app/Http/Kernel.php ဖိုင်မှာ အောက်ပါအတိုင်း middleware ကို register လုပ်ပါမယ်။

Code:

```
protected $routeMiddleware = [
    // ...တခြား middleware တွေ
    'check.admin' => \App\Http\Middleware\CheckAdmin::class,
];
```

✓ Route တွေ သတ်မှတ်ခြင်း

■ routes/web.php ဖိုင်မှာ အောက်ပါ route တွေကို ထည့်သွင်းပါမယ်။

Code:

```
<?php

use Illuminate\Support\Facades\Route;

// Authentication Routes
Route::get('/login', [LoginController::class, 'showLoginForm'])->name('logi
n');
Route::post('/login', [LoginController::class, 'login']);
```

```
// Admin Dashboard
Route::get('/admin/dashboard', function () {
    return view('admin.dashboard');
})->name('admin.dashboard')->middleware(['auth', 'check.admin']);

// User Dashboard
Route::get('/user/dashboard', function () {
    return view('user.dashboard');
})->name('user.dashboard')->middleware(['auth', 'check.admin']);

// Home Page
Route::get('/', function () {
    return view('welcome');
});
```

✓ User Model ပြင်ဆင်ခြင်း

■ User model မှာ usertype column အတွက် ပြင်ဆင်ပါမယ်။ app/Models/User.php

Code:

```
<?php

namespace App\Models;

use Illuminate\Foundation\Auth\User as Authenticatable;

class User extends Authenticatable
{
    protected $fillable = [
        'name',
        'email',
        'password',
        'usertype' // admin or user
    ];

    protected $hidden = [
        'password',
        'remember_token',
    ];
}
```

✓ Migration ဖိုင်ပြင်ဆင်ခြင်း

■ User table migration ဖိုင်ကို ပြင်ဆင်ပါမယ်။

Code:

```
public function up()
{
    Schema::create('users', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->string('email')->unique();
        $table->string('password');
        $table->enum('usertype', ['admin', 'user'])->default('user');
        $table->rememberToken();
        $table->timestamps();
    });
}
```

■ View Files တွေ ဖန်တီးခြင်း

resources/views/admin/dashboard.blade.php ဖိုင်ကို ဖန်တီးပါမယ်။

Code:

```
@extends('layouts.app')

@section('content')
<div class="container">
    <h1>Admin Dashboard</h1>
    <p>Welcome, Admin!</p>
</div>
@endsection
```

■ resources/views/user/dashboard.blade.php ဖိုင်ကို ဖန်တီးပါမယ်။

Code:

```
@extends('layouts.app')

@section('content')
```

```
<div class="container">
    <h1>User Dashboard</h1>
    <p>Welcome, User!</p>
</div>
@endsection
```

✓ အသုံးပြုပုံ ရှင်းလင်းချက်

→ User တစ်ယောက်က login ဝင်တဲ့အခါ CheckAdmin middleware က သူ့ရဲ့ usertype ကို စစ်ဆေးပါမယ်။

→ usertype က admin ဖြစ်နေရင် /admin/dashboard ကို redirect လုပ်ပေးပါမယ်။

→ usertype က user ဖြစ်နေရင် /user/dashboard ကို redirect လုပ်ပေးပါမယ်။

→ User က login မဝင်ရသေးရင် login page ကို redirect လုပ်ပေးပါမယ်။

✓ အထူးသတိပြုရန်

→ Database မှာ users table ထဲမှာ usertype column ပါရှိဖို့လိုပါတယ်။

→ User တစ်ယောက်ကို admin အဖြစ် သတ်မှတ်ဖို့ database မှာ usertype ကို admin လို့ update လုပ်ရပါမယ်။

→ Middleware ကို route တွေမှာ မှန်ကန်စွာ apply လုပ်ထားဖို့လိုပါတယ်။

✓ နောက်ထပ်တိုးတက်နိုင်သည့်အချက်များ

→ Admin နဲ့ user တွေအတွက် သီးသန့် controller တွေ ဖန်တီးနိုင်ပါတယ်။

→ Permission နဲ့ role management system တွေ ထပ်မံထည့်သွင်းနိုင်ပါတယ်။

→ Middleware ထဲမှာ ပိုမိုတိကျတဲ့ authorization logic တွေ ထည့်သွင်းနိုင်ပါတယ်။

ROLES & PERMISSIONS

Roles & Permissions

Laravel မှာ Roles and Permissions ကို အောက်ပါ package တွေသုံးပြီး အကောင်အထည် ဖော်လေ့ရှိ ပါတယ်။

✓ spatie/laravel-permission (အသုံးအများဆုံး)

ဒီ package က Laravel မှာ Roles and Permissions စနစ်ကို အလွယ်တကူ အကောင်အထည်ဖော်နိုင်ပါတယ်။

✓ အခြေခံအလုပ်လုပ်ပုံ

- Role - User တစ်ဦးရဲ့အခန်းကဏ္ဍ (ဥပမာ - Admin, Editor, User)
- Permission - User ဘာလုပ်ခွင့်ရှိလဲ (ဥပမာ - create-post, delete-post)

✓ အသုံးပြုပုံ အဆင့်ဆင့်

⇨ Installation

Command:

```
composer require spatie/laravel-permission
php artisan vendor:publish --provider="Spatie\Permission\PermissionServiceP
rovider"
php artisan migrate
```

⇨ Model မှာ Setup လုပ်ခြင်း

Code:

```
use Spatie\Permission\Traits\HasRoles;

class User extends Authenticatable
{
    use HasRoles;
    // ...
}
```

⇨ Role & Permission ဖန်တီးခြင်း

Code:

```
use Spatie\Permission\Models\Role;
use Spatie\Permission\Models\Permission;

// Permission ဖန်တီးခြင်း
$permission = Permission::create(['name' => 'edit-post']);

// Role ဖန်တီးပြီး Permission ပေးခြင်း
$role = Role::create(['name' => 'admin']);
$role->givePermissionTo('edit-post');
```

⇨ User ကို Role ပေးခြင်း

Code:

```
$user = User::find(1);
$user->assignRole('admin'); // Role ပေးခြင်း
$user->givePermissionTo('delete-post'); // တိုက်ရိုက် Permission ပေးခြင်း
```

⇨ Check Permissions

Code:

```
if ($user->hasRole('admin')) {
    // Admin ဖြစ်ပါက
}

if ($user->can('edit-post')) {
    // edit-post လုပ်ခွင့်ရှိပါက
}
```

⇨ Middleware အသုံးပြုခြင်း

Code:

```
Route::group(['middleware' => ['role:admin']], function () {
    Route::get('/admin', 'AdminController@index');
});
```

```
Route::group(['middleware' => ['permission:edit-post']], function () {
    Route::post('/posts/{id}', 'PostController@update');
});
```

✓ Laravel's Built-in Gates & Policies

Laravel မှာ Native Support အနေနဲ့ Gates နဲ့ Policies တွေကိုလည်း သုံးလို့ရပါတယ်။

■ Gates

Code:

```
// AuthServiceProvider.php မှာ Define လုပ်ပါ
Gate::define('edit-post', function ($user, $post) {
    return $user->id === $post->user_id;
});

// Controller/Blade မှာ Check လုပ်ပါ
if (Gate::allows('edit-post', $post)) {
    // လုပ်ဆောင်ချက်
}
```

■ Policies

Command:

```
php artisan make:policy PostPolicy --model=Post
```

Code:

```
class PostPolicy
{
    public function update(User $user, Post $post)
    {
        return $user->id === $post->user_id;
    }
}
```

✓ ဘယ်ဟာသုံးသင့်လဲ?

- အကြီးစား Project (Role-Based Access) → spatie/laravel-permission
- သေးငယ်ပြီး Simple Check → Gates & Policies

Laravel မှာ Roles and Permissions ကို ဒီလိုနည်းလမ်းတွေနဲ့ ထိရောက်စွာ အကောင်အထည်ဖော်နိုင်ပါတယ်။

လေ့ကျင့်ရန် -

Project structure:

```
role-permission-app/
├── app/
│   ├── Console/
│   ├── Exceptions/
│   ├── Http/
│   │   ├── Controllers/
│   │   │   ├── Auth/
│   │   │   ├── ProductController.php
│   │   │   ├── RoleController.php
│   │   │   ├── UserController.php
│   │   │   └── HomeController.php
│   │   ├── Middleware/
│   │   └── Requests/
│   ├── Models/
│   │   ├── User.php
│   │   ├── Role.php (Spatie package ကနေ extend)
│   │   ├── Permission.php (Spatie package ကနေ extend)
│   │   └── Product.php
│   └── Providers/
├── bootstrap/
├── config/
├── database/
│   ├── migrations/
│   │   ├── 2014_10_12_000000_create_users_table.php
│   │   ├── 2014_10_12_100000_create_password_resets_table.php
│   │   ├── 2020_01_01_000000_create_roles_table.php (Spatie)
│   │   ├── 2020_01_01_000000_create_permissions_table.php (Spatie)
```

```
|    |    ├── 2020_01_01_000000_create_model_has_permissions_table.php (Spatie)
|    |    ├── 2020_01_01_000000_create_model_has_roles_table.php (Spatie)
|    |    ├── 2020_01_01_000000_create_role_has_permissions_table.php (Spatie)
|    |    └── 2020_01_01_000000_create_products_table.php
|    └── seeders/
|        ├── DatabaseSeeder.php
|        └── RolePermissionSeeder.php
├── public/
├── resources/
|    ├── js/
|    ├── lang/
|    ├── sass/
|    └── views/
|        ├── auth/
|        ├── layouts/
|        |    └── app.blade.php
|        ├── products/
|        |    ├── index.blade.php
|        |    ├── create.blade.php
|        |    ├── edit.blade.php
|        |    └── show.blade.php
|        ├── roles/
|        |    ├── index.blade.php
|        |    ├── create.blade.php
|        |    └── edit.blade.php
|        ├── users/
|        |    ├── index.blade.php
|        |    ├── create.blade.php
|        |    ├── edit.blade.php
|        |    └── show.blade.php
|        └── home.blade.php
├── routes/
|    ├── api.php
|    ├── channels.php
|    ├── console.php
|    └── web.php
├── storage/
├── tests/
├── vendor/
├── .env
├── .env.example
├── artisan
├── composer.json
├── package.json
└── webpack.mix.js
```

✓အရေးကြီးသော File တွေရဲ့အသေးစိတ်

1. Controllers

Code:

```
app/Http/Controllers/
├─── ProductController.php - Product CRUD operations
├─── RoleController.php - Role management
└─── UserController.php - User management with role assignment
```

2. Models

Code:

```
app/Models/
├─── Product.php - Product model with fillable fields
└─── User.php - User model with HasRoles trait
```

3. Migrations

Code:

```
database/migrations/
├─── create_users_table.php - Users table
├─── create_products_table.php - Products table
└─── Spatie package migrations (5 files)
```

4. Views Structure

Code:

```
resources/views/
├─── layouts/
│    └─── app.blade.php - Main layout
├─── products/ - All product views
├─── roles/ - All role management views
└─── users/ - All user management views
```

5. Routes

Copy

Download

```
routes/web.php - Contains all web routes with middleware protection
```

6. Seeders

Copy

Download

```
database/seeders/RolePermissionSeeder.php - Initial roles and permissions
```

ဒီ structure က Laravel ရဲ့ standard structure ကိုအခြေခံပြီး roles and permissions system တစ်ခုအတွက် သင့်တော်တဲ့ ဖွဲ့စည်းပုံဖြစ်ပါတယ်။ Project ကြီးလာတာနဲ့အမျှ နောက်ထပ် folders တွေ ထပ်ဖွဲ့ဖို့ လိုအပ်လာနိုင်ပါတယ်။

☑ routes/web.php

ဒီ routes/web.php file မှာ User Management, Role Management နဲ့ Product Management တို့အတွက် routes တွေကို ဖြစ်ပါတယ်။

Code:

```
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\HomeController;
use App\Http\Controllers\UserController;
use App\Http\Controllers\RoleController;
use App\Http\Controllers\ProductController;

/*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/

// Authentication Routes (Laravel Default)
Auth::routes();

// Public Routes
Route::get('/', function () {
```

```
    return view('welcome');
});

// Authenticated Users Routes Group
Route::middleware(['auth'])->group(function () {

    // Dashboard Route
    Route::get('/home', [HomeController::class, 'index'])->name('home');

    // User Management Routes
    Route::prefix('users')->group(function () {
        Route::get('/', [UserController::class, 'index'])
            ->name('users.index')
            ->middleware('permission:manage-users');

        Route::get('/create', [UserController::class, 'create'])
            ->name('users.create')
            ->middleware('permission:manage-users');

        Route::post('/', [UserController::class, 'store'])
            ->name('users.store')
            ->middleware('permission:manage-users');

        Route::get('/{user}', [UserController::class, 'show'])
            ->name('users.show')
            ->middleware('permission:manage-users');

        Route::get('/{user}/edit', [UserController::class, 'edit'])
            ->name('users.edit')
            ->middleware('permission:manage-users');

        Route::put('/{user}', [UserController::class, 'update'])
            ->name('users.update')
            ->middleware('permission:manage-users');

        Route::delete('/{user}', [UserController::class, 'destroy'])
            ->name('users.destroy')
            ->middleware('permission:manage-users');
    });

    // Role Management Routes
    Route::prefix('roles')->group(function () {
        Route::get('/', [RoleController::class, 'index'])
            ->name('roles.index')
            ->middleware('permission:manage-users');
```

```
        Route::get('/create', [RoleController::class, 'create'])
            ->name('roles.create')
            ->middleware('permission:manage-users');

        Route::post('/', [RoleController::class, 'store'])
            ->name('roles.store')
            ->middleware('permission:manage-users');

        Route::get('/{role}/edit', [RoleController::class, 'edit'])
            ->name('roles.edit')
            ->middleware('permission:manage-users');

        Route::put('/{role}', [RoleController::class, 'update'])
            ->name('roles.update')
            ->middleware('permission:manage-users');

        Route::delete('/{role}', [RoleController::class, 'destroy'])
            ->name('roles.destroy')
            ->middleware('permission:manage-users');
    });

    // Product Management Routes
    Route::prefix('products')->group(function () {
        Route::get('/', [ProductController::class, 'index'])
            ->name('products.index')
            ->middleware('permission:manage-products|view-products');

        Route::get('/create', [ProductController::class, 'create'])
            ->name('products.create')
            ->middleware('permission:manage-products');

        Route::post('/', [ProductController::class, 'store'])
            ->name('products.store')
            ->middleware('permission:manage-products');

        Route::get('/{product}', [ProductController::class, 'show'])
            ->name('products.show')
            ->middleware('permission:view-products');

        Route::get('/{product}/edit', [ProductController::class, 'edit'])
            ->name('products.edit')
            ->middleware('permission:manage-products');

        Route::put('/{product}', [ProductController::class, 'update'])
            ->name('products.update')
            ->middleware('permission:manage-products');
```

```
        Route::delete('/{product}', [ProductController::class, 'destroy'])
            ->name('products.destroy')
            ->middleware('permission:manage-products');
    });

    // Admin Only Area
    Route::middleware(['role:admin'])->group(function () {
        Route::get('/admin-dashboard', function () {
            return view('admin.dashboard');
        })->name('admin.dashboard');
    });
});

// Fallback Route
Route::fallback(function () {
    return view('errors.404');
});
```

✓ ရှင်းလင်းချက်

1. Authentication Routes
   Auth::routes() က Laravel ရဲ့ default auth routes တွေကို register လုပ်ပေးပါတယ်။
2. Public Routes
   / route မှာ welcome page ကို ပြသပါတယ်။
3. Authenticated Routes Group
   auth middleware နဲ့ အကုန်ဝင်ရမယ့် routes တွေကို group လုပ်ထားပါတယ်။
4. Permission Protection
   - User management routes တွေမှာ manage-users permission လိုပါတယ်။
   - Role management routes တွေမှာ manage-users permission လိုပါတယ်။
   - Product management routes တွေမှာ manage-products ဒါမှမဟုတ် view-products permission တစ်ခုခုလိုပါတယ်။
5. Role-Based Access
   /admin-dashboard route က admin role ရှိသူတွေသာ ဝင်ရောက်နိုင်ပါတယ်။
6. Fallback Route
   မရှိတဲ့ route တွေအတွက် 404 page ပြသပါတယ်။

ဒီ route setup ကို သုံးပြီး သင့် Laravel application မှာ role-based access control system ကို အလွယ်တကူ implement လုပ်နိုင်ပါတယ်။

☑ Migrations

ဒီအပိုင်းမှာတော့ ကျွန်တော်တို့ရဲ့ Laravel Roles and Permissions App အတွက် migration files တွေကို ရေးပြပေးပါမယ်။

✓ Users Table Migration
(2014_10_12_000000_create_users_table.php)

Code:

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('email')->unique();
            $table->timestamp('email_verified_at')->nullable();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::dropIfExists('users');
    }
};
```

✓ Password Resets Table Migration
(2014_10_12_100000_create_password_resets_table.php)

Code:

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        Schema::create('password_resets', function (Blueprint $table) {
            $table->string('email')->index();
            $table->string('token');
            $table->timestamp('created_at')->nullable();
        });
    }

    public function down()
    {
        Schema::dropIfExists('password_resets');
    }
};
```

✓ Products Table Migration

(2023_01_01_000000_create_products_table.php)

Code:

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        Schema::create('products', function (Blueprint $table) {
```

```
            $table->id();
            $table->string('name');
            $table->text('description');
            $table->decimal('price', 10, 2);
            $table->timestamps();
            $table->softDeletes();
        });
    }

    public function down()
    {
        Schema::dropIfExists('products');
    }
};
```

✓ Spatie Permission Package Migrations

⇨ Roles Table (2023_01_01_000000_create_roles_table.php)

Code:

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        $tableNames = config('permission.table_names');

        Schema::create($tableNames['roles'], function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->string('name')->unique();
            $table->string('guard_name')->default('web');
            $table->timestamps();
        });
    }

    public function down()
    {
        $tableNames = config('permission.table_names');
```

```
        Schema::drop($tableNames['roles']);
    }
};
```

⇨ Permissions Table
(2023_01_01_000000_create_permissions_table.php)

Code:

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        $tableNames = config('permission.table_names');

        Schema::create($tableNames['permissions'], function (Blueprint $tab
le) {
            $table->bigIncrements('id');
            $table->string('name')->unique();
            $table->string('guard_name')->default('web');
            $table->timestamps();
        });
    }

    public function down()
    {
        $tableNames = config('permission.table_names');
        Schema::drop($tableNames['permissions']);
    }
};
```

⇨ Model Has Permissions Table
(2023_01_01_000000_create_model_has_permissions_table.php)

Code:

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        $tableNames = config('permission.table_names');
        $columnNames = config('permission.column_names');

        Schema::create($tableNames['model_has_permissions'], function (Blue
print $table) use ($tableNames, $columnNames) {
            $table->unsignedBigInteger('permission_id');

            $table->string('model_type');
            $table->unsignedBigInteger($columnNames['model_morph_key']);
            $table->index([$columnNames['model_morph_key'], 'model_type'],
'model_has_permissions_model_id_model_type_index');

            $table->foreign('permission_id')
                ->references('id')
                ->on($tableNames['permissions'])
                ->onDelete('cascade');

            $table->primary(['permission_id', $columnNames['model_morph_key
'], 'model_type'],
                'model_has_permissions_permission_model_type_primary');
        });
    }

    public function down()
    {
        $tableNames = config('permission.table_names');
        Schema::drop($tableNames['model_has_permissions']);
    }
};
```

⇨ Model Has Roles Table
(2023_01_01_000000_create_model_has_roles_table.php)

Code:

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        $tableNames = config('permission.table_names');
        $columnNames = config('permission.column_names');

        Schema::create($tableNames['model_has_roles'], function (Blueprint
$table) use ($tableNames, $columnNames) {
            $table->unsignedBigInteger('role_id');

            $table->string('model_type');
            $table->unsignedBigInteger($columnNames['model_morph_key']);
            $table->index([$columnNames['model_morph_key'], 'model_type'],
'model_has_roles_model_id_model_type_index');

            $table->foreign('role_id')
                ->references('id')
                ->on($tableNames['roles'])
                ->onDelete('cascade');

            $table->primary(['role_id', $columnNames['model_morph_key'], 'm
odel_type'],
                'model_has_roles_role_model_type_primary');
        });
    }

    public function down()
    {
        $tableNames = config('permission.table_names');
```

```
        Schema::drop($tableNames['model_has_roles']);
    }
};
```

⇨ Role Has Permissions Table
(2023_01_01_000000_create_role_has_permissions_table.php)

Code:

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        $tableNames = config('permission.table_names');

        Schema::create($tableNames['role_has_permissions'], function (Bluep
rint $table) use ($tableNames) {
            $table->unsignedBigInteger('permission_id');
            $table->unsignedBigInteger('role_id');

            $table->foreign('permission_id')
                ->references('id')
                ->on($tableNames['permissions'])
                ->onDelete('cascade');

            $table->foreign('role_id')
                ->references('id')
                ->on($tableNames['roles'])
                ->onDelete('cascade');

            $table->primary(['permission_id', 'role_id'], 'role_has_permiss
ions_permission_id_role_id_primary');
        });
    }

    public function down()
```

```
    {
        $tableNames = config('permission.table_names');
        Schema::drop($tableNames['role_has_permissions']);
    }
};
```

✓ Migration တွေရဲ့အရေးကြီးတဲ့ အပိုင်းများ

1. Users Table မှာ:
    - အခြေခံ user information တွေကို သိမ်းဆည်းဖို့ ဖြစ်ပါတယ်
    - email column ကို unique လုပ်ထားပါတယ်
2. Products Table မှာ:
    - Product နဲ့ဆိုင်တဲ့ data တွေကို သိမ်းဆည်းဖို့ ဖြစ်ပါတယ်
    - softDeletes() ကို သုံးထားတာကြောင့် product တွေကို restore လုပ်နိုင်ပါတယ်
3. Spatie Permission Migrations မှာ:
    - Role နဲ့ Permission management အတွက် လိုအပ်တဲ့ tables တွေကို ဖန်တီးပေးပါတယ်
    - Many-to-many relationships တွေကို သတ်မှတ်ထားပါတယ်
    - Foreign key constraints တွေကို သတ်မှတ်ထားပါတယ်

ဒီ migration files တွေကို run ပြီးရင် database structure က အောက်ပါအတိုင်းဖြစ်ပါတယ်:

Model Data Strucutre:

```
users
  - id
  - name
  - email
  - password
  - remember_token
  - created_at
  - updated_at

products
  - id
  - name
  - description
  - price
```

```
  - created_at
  - updated_at
  - deleted_at

roles
  - id
  - name
  - guard_name
  - created_at
  - updated_at

permissions
  - id
  - name
  - guard_name
  - created_at
  - updated_at

model_has_permissions
  - permission_id
  - model_type
  - model_id

model_has_roles
  - role_id
  - model_type
  - model_id

role_has_permissions
  - permission_id
  - role_id
```

ဒီ migration တွေကို run ဖို့အတွက် terminal မှာ အောက်ပါ command ကို run ပေးပါ:

Command:

```
php artisan migrate
```

☑ Models

Roles and Permissions App အတွက် Model တွေကို ရေးပြပေးပါမယ်။ ဖြစ်ပါတယ်၊ မယ်၊ မှာ စတဲ့ စကားလုံးတွေကို ပြင်ရေးပေးထားပါတယ်။

✓ User Model (app/Models/User.php)

Code:

```
<?php

namespace App\Models;

use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use Spatie\Permission\Traits\HasRoles;
use Laravel\Sanctum\HasApiTokens;

class User extends Authenticatable
{
    use HasApiTokens, HasFactory, Notifiable, HasRoles;

    /**
     * The attributes that are mass assignable.
     *
     * @var array<int, string>
     */
    protected $fillable = [
        'name',
        'email',
        'password',
    ];

    /**
     * The attributes that should be hidden for serialization.
     *
     * @var array<int, string>
     */
    protected $hidden = [
        'password',
        'remember_token',
```

```
    ];

    /**
     * The attributes that should be cast.
     *
     * @var array<string, string>
     */
    protected $casts = [
        'email_verified_at' => 'datetime',
    ];

    /**
     * ကိုယ်ပိုင်method တွေ ထပ်ဖြည့်ချင်ရင်
     */
    public function isAdmin()
    {
        return $this->hasRole('admin');
    }

    public function getRoleNamesAttribute()
    {
        return $this->roles->pluck('name')->implode(', ');
    }
}
```

✓ Product Model (app/Models/Product.php)

Code:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Product extends Model
{
    use HasFactory;

    /**
     * The attributes that are mass assignable.
     *
     * @var array<int, string>
     */
```

```
    protected $fillable = [
        'name',
        'description',
        'price',
    ];

    /**
     * The attributes that should be cast.
     *
     * @var array<string, string>
     */
    protected $casts = [
        'price' => 'decimal:2',
    ];

    /**
     * Accessor for formatted price
     */
    public function getFormattedPriceAttribute()
    {
        return '₱'.number_format($this->price, 2);
    }

    /**
     * Scope for expensive products
     */
    public function scopeExpensive($query)
    {
        return $query->where('price', '>', 1000);
    }
}
```

✓ Role Model (Spatie Package ကနေ Extend လုပ်ထားတာ)

Code:

```
<?php

namespace Spatie\Permission\Models;

use Illuminate\Database\Eloquent\Relations\BelongsToMany;
use Spatie\Permission\Contracts\Role as RoleContract;
use Spatie\Permission\Exceptions\GuardDoesNotMatch;
```

```
use Spatie\Permission\Exceptions\RoleAlreadyExists;
use Spatie\Permission\Exceptions\RoleDoesNotExist;
use Spatie\Permission\Guard;
use Spatie\Permission\Traits\HasPermissions;
use Spatie\Permission\Traits\RefreshesPermissionCache;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\MorphToMany;

class Role extends Model implements RoleContract
{
    use HasPermissions;
    use RefreshesPermissionCache;

    protected $guarded = ['id'];

    public function __construct(array $attributes = [])
    {
        $attributes['guard_name'] = $attributes['guard_name'] ?? config('au
th.defaults.guard');

        parent::__construct($attributes);

        $this->setTable(config('permission.table_names.roles'));
    }

    public static function create(array $attributes = [])
    {
        $attributes['guard_name'] = $attributes['guard_name'] ?? Guard::get
DefaultName(static::class);

        if (static::where('name', $attributes['name'])->where('guard_name',
$attributes['guard_name'])->first()) {
            throw RoleAlreadyExists::create($attributes['name'], $attribute
s['guard_name']);
        }

        return static::query()->create($attributes);
    }

    public static function findByName(string $name, $guardName = null): Rol
eContract
    {
        $guardName = $guardName ?? Guard::getDefaultName(static::class);

        $role = static::where('name', $name)->where('guard_name', $guardNam
e)->first();
```

```
        if (! $role) {
            throw RoleDoesNotExist::named($name);
        }

        return $role;
    }

    public static function findById(int $id, $guardName = null): RoleContra
ct
    {
        $guardName = $guardName ?? Guard::getDefaultName(static::class);

        $role = static::where('id', $id)->where('guard_name', $guardName)->
first();

        if (! $role) {
            throw RoleDoesNotExist::withId($id);
        }

        return $role;
    }

    public static function findOrCreate(string $name, $guardName = null): R
oleContract
    {
        $guardName = $guardName ?? Guard::getDefaultName(static::class);

        $role = static::where('name', $name)->where('guard_name', $guardNam
e)->first();

        if (! $role) {
            return static::query()->create(['name' => $name, 'guard_name' =
> $guardName]);
        }

        return $role;
    }

    public function hasPermissionTo($permission): bool
    {
        $permissionClass = $this->getPermissionClass();

        if (is_string($permission)) {
            $permission = $permissionClass->findByName($permission, $this->
guard_name);
```

```
        }

        if (is_int($permission)) {
            $permission = $permissionClass->findById($permission, $this->gu
ard_name);
        }

        if (! $this->getGuardNames()->contains($permission->guard_name)) {
            throw GuardDoesNotMatch::create($permission->guard_name, $this-
>getGuardNames());
        }

        return $this->permissions->contains('id', $permission->id);
    }
}
```

✓ Permission Model (Spatie Package ကနေ Extend လုပ်ထားတာ)

Code:

```
<?php

namespace Spatie\Permission\Models;

use Spatie\Permission\Contracts\Permission as PermissionContract;
use Spatie\Permission\Exceptions\PermissionAlreadyExists;
use Spatie\Permission\Exceptions\PermissionDoesNotExist;
use Spatie\Permission\Guard;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\MorphToMany;

class Permission extends Model implements PermissionContract
{
    protected $guarded = ['id'];

    public function __construct(array $attributes = [])
    {
        $attributes['guard_name'] = $attributes['guard_name'] ?? config('au
th.defaults.guard');

        parent::__construct($attributes);

        $this->setTable(config('permission.table_names.permissions'));
    }
```

```
    public static function create(array $attributes = [])
    {
        $attributes['guard_name'] = $attributes['guard_name'] ?? Guard::get
DefaultName(static::class);

        if (static::where('name', $attributes['name'])->where('guard_name',
$attributes['guard_name'])->first()) {
            throw PermissionAlreadyExists::create($attributes['name'], $att
ributes['guard_name']);
        }

        return static::query()->create($attributes);
    }

    public static function findByName(string $name, $guardName = null): Per
missionContract
    {
        $guardName = $guardName ?? Guard::getDefaultName(static::class);

        $permission = static::where('name', $name)->where('guard_name', $gu
ardName)->first();

        if (! $permission) {
            throw PermissionDoesNotExist::create($name, $guardName);
        }

        return $permission;
    }

    public static function findById(int $id, $guardName = null): Permission
Contract
    {
        $guardName = $guardName ?? Guard::getDefaultName(static::class);

        $permission = static::where('id', $id)->where('guard_name', $guardN
ame)->first();

        if (! $permission) {
            throw PermissionDoesNotExist::withId($id, $guardName);
        }

        return $permission;
    }

    public static function findOrCreate(string $name, $guardName = null): P
ermissionContract
```

```
    {
        $guardName = $guardName ?? Guard::getDefaultName(static::class);

        $permission = static::where('name', $name)->where('guard_name', $gu
ardName)->first();

        if (! $permission) {
            return static::query()->create(['name' => $name, 'guard_name' =
> $guardName]);
        }

        return $permission;
    }

    public function roles(): MorphToMany
    {
        return $this->morphedByMany(
            config('permission.models.role'),
            'model',
            config('permission.table_names.model_has_permissions'),
            'permission_id',
            'model_id'
        );
    }

    public function users(): MorphToMany
    {
        return $this->morphedByMany(
            getModelForGuard($this->attributes['guard_name']),
            'model',
            config('permission.table_names.model_has_permissions'),
            'permission_id',
            'model_id'
        );
    }
}
```

✓ Model တွေရဲ့အရေးကြီးတဲ့ အပိုင်းများ

1. User Model မှာ:

- HasRoles trait ကို သုံးထားတာကြောင့် role နဲ့ permission တွေကို manage လုပ်နိုင်ပါတယ်
- isAdmin() method နဲ့ admin role ရှိမရှိ check လုပ်နိုင်ပါတယ်
- getRoleNamesAttribute accessor နဲ့ user ရဲ့ role names တွေကို comma separated string အဖြစ် ရနိုင်ပါတယ်

2. Product Model မှာ:
   - fillable properties တွေကို သတ်မှတ်ထားပါတယ်
   - getFormattedPriceAttribute accessor နဲ့ price ကို formatted လုပ်ပြီး ပြနိုင်ပါတယ်
   - scopeExpensive local scope နဲ့ expensive products တွေကို filter လုပ်နိုင်ပါတယ်
3. Role & Permission Models:
   - Spatie package ကနေ extend လုပ်ထားတာဖြစ်ပါတယ်
   - Role နဲ့ Permission တွေကို manage လုပ်ဖို့ method တွေ အပြည့်အစုံပါဝင်ပါတယ်
   - Database table names တွေကို config ကနေ ယူသုံးထားပါတယ်

ဒီ Model တွေကို အသုံးပြုပြီး သင့် Laravel application မှာ role-based access control system ကို အကောင်အထည်ဖော်နိုင်ပါတယ်။

☑ Controllers

Laravel Roles and Permissions Controllers

User, Role နဲ့ Product management တို့အတွက် controller codes တွေကို ရေးပေးပါမယ်။

✓ UserController.php

Code:

```
<?php
```

```
namespace App\Http\Controllers;

use App\Models\User;
use App\Models\Role;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Hash;
use Illuminate\Validation\Rules;

class UserController extends Controller
{
    public function __construct()
    {
        $this->middleware('permission:manage-users');
    }

    public function index()
    {
        $users = User::with('roles')->paginate(10);
        return view('users.index', compact('users'));
    }

    public function create()
    {
        $roles = Role::all();
        return view('users.create', compact('roles'));
    }

    public function store(Request $request)
    {
        $request->validate([
            'name' => 'required|string|max:255',
            'email' => 'required|string|email|max:255|unique:users',
            'password' => ['required', 'confirmed', Rules\Password::default
s()],
            'roles' => 'required|array',
        ]);

        $user = User::create([
            'name' => $request->name,
            'email' => $request->email,
            'password' => Hash::make($request->password),
        ]);

        $user->syncRoles($request->roles);
```

```
        return redirect()->route('users.index')
            ->with('success', 'User created successfully');
    }

    public function show(User $user)
    {
        return view('users.show', compact('user'));
    }

    public function edit(User $user)
    {
        $roles = Role::all();
        return view('users.edit', compact('user', 'roles'));
    }

    public function update(Request $request, User $user)
    {
        $request->validate([
            'name' => 'required|string|max:255',
            'email' => 'required|string|email|max:255|unique:users,email,'.
$user->id,
            'password' => ['nullable', 'confirmed', Rules\Password::default
s()],
            'roles' => 'required|array',
        ]);

        $user->update([
            'name' => $request->name,
            'email' => $request->email,
        ]);

        if ($request->password) {
            $user->update(['password' => Hash::make($request->password)]);
        }

        $user->syncRoles($request->roles);

        return redirect()->route('users.index')
            ->with('success', 'User updated successfully');
    }

    public function destroy(User $user)
    {
        $user->delete();
        return redirect()->route('users.index')
            ->with('success', 'User deleted successfully');
```

```
    }
}
```

✓ RoleController.php

Code:

```
<?php

namespace App\Http\Controllers;

use Spatie\Permission\Models\Role;
use Spatie\Permission\Models\Permission;
use Illuminate\Http\Request;

class RoleController extends Controller
{
    public function __construct()
    {
        $this->middleware('permission:manage-users');
    }

    public function index()
    {
        $roles = Role::with('permissions')->paginate(10);
        return view('roles.index', compact('roles'));
    }

    public function create()
    {
        $permissions = Permission::all();
        return view('roles.create', compact('permissions'));
    }

    public function store(Request $request)
    {
        $request->validate([
            'name' => 'required|string|unique:roles',
            'permissions' => 'required|array',
        ]);

        $role = Role::create(['name' => $request->name]);
        $role->syncPermissions($request->permissions);
```

```
        return redirect()->route('roles.index')
            ->with('success', 'Role created successfully');
    }

    public function edit(Role $role)
    {
        $permissions = Permission::all();
        return view('roles.edit', compact('role', 'permissions'));
    }

    public function update(Request $request, Role $role)
    {
        $request->validate([
            'name' => 'required|string|unique:roles,name,'.$role->id,
            'permissions' => 'required|array',
        ]);

        $role->update(['name' => $request->name]);
        $role->syncPermissions($request->permissions);

        return redirect()->route('roles.index')
            ->with('success', 'Role updated successfully');
    }

    public function destroy(Role $role)
    {
        $role->delete();
        return redirect()->route('roles.index')
            ->with('success', 'Role deleted successfully');
    }
}
```

✓ ProductController.php

Code:

```
<?php

namespace App\Http\Controllers;

use App\Models\Product;
use Illuminate\Http\Request;

class ProductController extends Controller
{
```

```
    public function __construct()
    {
        $this->middleware('permission:manage-products|view-products', ['exc
ept' => ['index', 'show']]);
        $this->middleware('permission:manage-products', ['except' => ['inde
x', 'show']]);
    }

    public function index()
    {
        $products = Product::latest()->paginate(10);
        return view('products.index', compact('products'));
    }

    public function create()
    {
        return view('products.create');
    }

    public function store(Request $request)
    {
        $request->validate([
            'name' => 'required|string|max:255',
            'description' => 'required|string',
            'price' => 'required|numeric|min:0',
        ]);

        Product::create($request->all());

        return redirect()->route('products.index')
            ->with('success', 'Product created successfully');
    }

    public function show(Product $product)
    {
        return view('products.show', compact('product'));
    }

    public function edit(Product $product)
    {
        return view('products.edit', compact('product'));
    }

    public function update(Request $request, Product $product)
    {
        $request->validate([
```

```
            'name' => 'required|string|max:255',
            'description' => 'required|string',
            'price' => 'required|numeric|min:0',
        ]);

        $product->update($request->all());

        return redirect()->route('products.index')
            ->with('success', 'Product updated successfully');
    }

    public function destroy(Product $product)
    {
        $product->delete();
        return redirect()->route('products.index')
            ->with('success', 'Product deleted successfully');
    }
}
```

✓ HomeController.php

Code:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class HomeController extends Controller
{
    public function __construct()
    {
        $this->middleware('auth');
    }

    public function index()
    {
        return view('home');
    }
}
```

✓ ရှင်းလင်းချက်

1. UserController မှာ:
    - User management အတွက် CRUD operations ပါဝင်ပါတယ်
    - Role assignment လုပ်နိုင်ပါတယ်
    - Password hashing ကို အလိုအလျောက်လုပ်ပေးပါတယ်
2. RoleController မှာ:
    - Role management အတွက် CRUD operations ပါဝင်ပါတယ်
    - Permission assignment လုပ်နိုင်ပါတယ်
    - Spatie package ကို အသုံးပြုထားပါတယ်
3. ProductController မှာ:
    - Product management အတွက် CRUD operations ပါဝင်ပါတယ်
    - View permission နဲ့ Manage permission ကို ခွဲခြားထားပါတယ်
4. HomeController မှာ:
    - Authenticated users အတွက် dashboard ကိုပြပေးပါတယ်

ဒီ controller တွေကို အသုံးပြုပြီး သင့် Laravel application မှာ role-based access control system ကို အကောင်အထည်ဖော်နိုင်ပါတယ်။

☑ Seeder

ဒီအပိုင်းမှာတော့ ကျွန်တော်တို့ရဲ့ Laravel Roles and Permissions App အတွက် seeder files တွေကို ရေးပြပေးပါမယ်။ ဖြစ်ပါတယ်၊ မယ်၊ မှာ စတဲ့ စကားလုံးတွေကို ပြင်ရေးပေးထားပါတယ်။

✓ Database Seeder (database/seeders/DatabaseSeeder.php)

Code:

```
<?php
```

```
namespace Database\Seeders;

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    public function run()
    {
        $this->call([
            RolePermissionSeeder::class,
            UserSeeder::class,
            ProductSeeder::class,
        ]);
    }
}
```

✓ Role & Permission Seeder (database/seeders/RolePermissionSeeder.php)

Code:

```
<?php

namespace Database\Seeders;

use Illuminate\Database\Seeder;
use Spatie\Permission\Models\Role;
use Spatie\Permission\Models\Permission;

class RolePermissionSeeder extends Seeder
{
    public function run()
    {
        // Permissions များ ဖန်တီးခြင်း
        $permissions = [
            'manage-users',
            'view-users',
            'create-users',
            'edit-users',
            'delete-users',

            'manage-roles',
            'view-roles',
            'create-roles',
            'edit-roles',
```

```
            'delete-roles',

            'manage-products',
            'view-products',
            'create-products',
            'edit-products',
            'delete-products',

            'manage-categories',
            'view-categories',
            'create-categories',
            'edit-categories',
            'delete-categories',
        ];

        foreach ($permissions as $permission) {
            Permission::create(['name' => $permission, 'guard_name' => 'web
']);
        }

        // Roles များ ဖန်တီးခြင်း
        $adminRole = Role::create(['name' => 'admin', 'guard_name' => 'web'
]);
        $editorRole = Role::create(['name' => 'editor', 'guard_name' => 'we
b']);
        $userRole = Role::create(['name' => 'user', 'guard_name' => 'web'])
;

        // Admin Role ကို Permissions များ ပေးခြင်း
        $adminRole->givePermissionTo([
            'manage-users',
            'manage-roles',
            'manage-products',
            'manage-categories',
        ]);

        // Editor Role ကို Permissions များ ပေးခြင်း
        $editorRole->givePermissionTo([
            'view-users',
            'manage-products',
            'view-products',
            'create-products',
            'edit-products',
            'delete-products',
        ]);
```

```
        // User Role ကို Permissions များ ပေးခြင်း
        $userRole->givePermissionTo([
            'view-products',
        ]);
    }
}
```

✓ User Seeder (database/seeders/UserSeeder.php)

Code:

```
<?php

namespace Database\Seeders;

use App\Models\User;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\Hash;

class UserSeeder extends Seeder
{
    public function run()
    {
        // Admin User ဖန်တီးခြင်း
        $admin = User::create([
            'name' => 'Admin User',
            'email' => 'admin@example.com',
            'password' => Hash::make('password'),
        ]);
        $admin->assignRole('admin');

        // Editor User ဖန်တီးခြင်း
        $editor = User::create([
            'name' => 'Editor User',
            'email' => 'editor@example.com',
            'password' => Hash::make('password'),
        ]);
        $editor->assignRole('editor');

        // Normal User ဖန်တီးခြင်း
        $user = User::create([
            'name' => 'Regular User',
```

```
            'email' => 'user@example.com',
            'password' => Hash::make('password'),
        ]);
        $user->assignRole('user');

        // Additional Users (Optional)
        User::factory()->count(10)->create()->each(function ($user) {
            $user->assignRole('user');
        });
    }
}
```

✓ Product Seeder (database/seeders/ProductSeeder.php)

Code:

```
<?php

namespace Database\Seeders;

use App\Models\Product;
use Illuminate\Database\Seeder;

class ProductSeeder extends Seeder
{
    public function run()
    {
        $products = [
            [
                'name' => 'Laptop',
                'description' => 'High performance laptop with 16GB RAM',
                'price' => 1200.00,
            ],
            [
                'name' => 'Smartphone',
                'description' => 'Latest smartphone with 5G support',
                'price' => 800.00,
            ],
            [
                'name' => 'Headphones',
                'description' => 'Noise cancelling wireless headphones',
                'price' => 250.00,
            ],
            [
```

```
                'name' => 'Smart Watch',
                'description' => 'Fitness tracking smart watch',
                'price' => 199.99,
            ],
            [
                'name' => 'Tablet',
                'description' => '10-inch tablet with stylus support',
                'price' => 450.00,
            ],
        ];

        foreach ($products as $product) {
            Product::create($product);
        }

        // Optional: Factory သုံးပြီး Random Products များများ ဖန်တီးခြင်း
        Product::factory()->count(15)->create();
    }
}
```

✓ Seeder တွေကို အသုံးပြုနည်း

Seeder တွေကို run ဖို့အတွက် အောက်ပါ command တွေကို အသုံးပြုနိုင်ပါတယ်:

1. အကုန်လုံးကို တစ်ခါတည်း run ဖို့ command:

Command:

```
php artisan db:seed
```

2. တစ်ခုချင်းစီ run ချင်ရင် သုံးရမယ့် command:

Command:

```
php artisan db:seed --class=RolePermissionSeeder
php artisan db:seed --class=UserSeeder
php artisan db:seed --class=ProductSeeder
```

3. Migration တွေနဲ့အတူတူ run ဖို့ command:

Command:

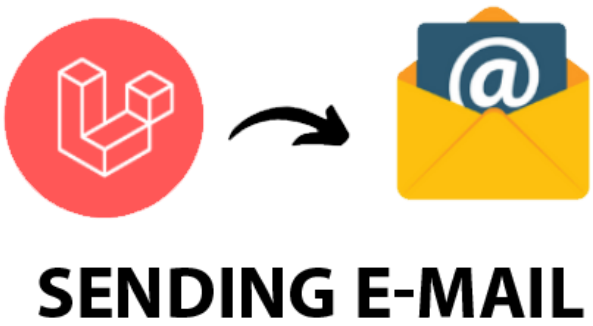
```
php artisan migrate:fresh --seed
```

✓ ရှင်းလင်းချက်

1. RolePermissionSeeder မှာ:
    - အခြေခံ permission တွေနဲ့ role တွေကို ဖန်တီးပါတယ်
    - Role တစ်ခုချင်းစီကို သက်ဆိုင်ရာ permission တွေ assign လုပ်ပါတယ်
2. UserSeeder မှာ:
    - Default users (admin, editor, user) တွေကို ဖန်တီးပါတယ်
    - User တစ်ဦးချင်းစီကို သက်ဆိုင်ရာ role တွေ assign လုပ်ပါတယ်
    - Optional အနေနဲ့ factory သုံးပြီး random users တွေ ဖန်တီးနိုင်ပါတယ်
3. ProductSeeder မှာ:
    - Sample products တွေကို ဖန်တီးပါတယ်
    - Optional အနေနဲ့ factory သုံးပြီး random products တွေ ဖန်တီးနိုင်ပါတယ်

ဒီ seeder တွေကို အသုံးပြုပြီး development environment မှာ testing အတွက် initial data တွေကို အလွယ်တကူ ထည့်သွင်းနိုင်ပါတယ်။

Sending Email



Sending Email

Laravel မှာ Email ပို့တဲ့ လုပ်ဆောင်ချက်ကို အလွယ်တကူ အသုံးပြုနိုင်အောင် Mail Facade နဲ့ Mailable Class တို့ကို support ပေးထားတယ်။ Gmail, SMTP, Mailgun, Sendmail စတဲ့ Email Service တွေနဲ့လည်း အလွယ်တကူ ချိတ်ဆက်အသုံးပြုနိုင်ပါတယ်။

✓ Email ပို့ဖို့ ပြင်ဆင်ခြင်း

Laravel မှာ Email ပို့ဖို့အတွက် `.env` ဖိုင်မှာ Email Setting တွေကို အောက်ပါအတိုင်း သတ်မှတ်ရမယ်။

Config:

```
MAIL_MAILER=smtp
MAIL_HOST=smtp.gmail.com
MAIL_PORT=587
MAIL_USERNAME=your@gmail.com
MAIL_PASSWORD=your-app-password (Gmail App Password)
MAIL_ENCRYPTION=tls
MAIL_FROM_ADDRESS=your@gmail.com
MAIL_FROM_NAME="${APP_NAME}"
```

✍ မှတ်ချက်: Gmail သုံးမယ်ဆိုရင် "Less Secure Apps" ကို Enable လုပ်ဖို့ (သို့) App Password ယူဖို့ လိုမယ်။

✓ Mailable Class ဖန်တီးခြင်း

Laravel မှာ Email Template တွေကို Manage လုပ်ဖို့ **Mailable Class** ကို အသုံးပြုတယ်။ အောက်ပါ Command နဲ့ Mailable Class ဖန်တီးလို့ရတယ်။

Command:

```
php artisan make:mail WelcomeEmail
```

ဒီ Command က app/Mail/WelcomeEmail.php ဆိုတဲ့ File တစ်ခုကို ဖန်တီးပေးမယ်။

Code:

```
<?php

namespace App\Mail;

use Illuminate\Bus\Queueable;
use Illuminate\Mail\Mailable;
use Illuminate\Queue\SerializesModels;

class WelcomeEmail extends Mailable
{
    use Queueable, SerializesModels;

    public $user;

    public function __construct($user)
    {
        $this->user = $user;
    }

    public function build()
    {
        return $this->subject('Welcome to My App')
                    ->view('emails.welcome'); // resources/views/emails/wel
come.blade.php
    }
}
```

✓ Email View (Blade Template) ဖန်တီးခြင်း

Email မှာ ပြမယ့် HTML Content တွေကို **Blade Template** အဖြစ် ဖန်တီးရမယ်။

Command:

```
mkdir -p resources/views/emails
touch resources/views/emails/welcome.blade.php
```

အောက်ပါအတိုင်း Email Template ကို ရေးလို့ရတယ်။

Code:

```
<!-- resources/views/emails/welcome.blade.php -->
<!DOCTYPE html>
<html>
<head>
    <title>Welcome Email</title>
</head>
<body>
    <h1>Hello, {{ $user->name }}!</h1>
    <p>Thanks for joining our application.</p>
</body>
</html>
```

---

4. Controller ကနေ Email ပို့ခြင်း

အခု WelcomeEmail ကို Controller ကနေ အောက်ပါအတိုင်း ပို့လို့ရတယ်။

Code:

```
use App\Mail\WelcomeEmail;
use Illuminate\Support\Facades\Mail;

public function sendWelcomeEmail()
{
    $user = User::find(1); // Example User
    Mail::to($user->email)->send(new WelcomeEmail($user));
    return "Email sent successfully!";
}
```

✓ Email Queue အသုံးပြုခြင်း (Optional)

Email တွေကို **Queue** နဲ့ ပို့မယ်ဆိုရင် Mail::queue() ကို သုံးလို့ရတယ်။

Code:

```
Mail::to($user->email)->queue(new WelcomeEmail($user));
```

Queue Worker ကို Start လုပ်ဖို့:

Comand:

```
php artisan queue:work
```

✓ Email Testing (Local Development)

Local Development အတွက် **Mailtrap** သို့မဟုတ် **Log Driver** ကို အသုံးပြုနိုင်ပါတယ်။

.env ဖိုင်မှာ ပြောင်းရန်:

Config:

```
MAIL_MAILER=log
```

ဒါဆိုရင် Email တွေကို storage/logs/laravel.log မှာ ကြည့်လို့ရမယ်။

✓ Gmail နဲ့ SMTP စမ်းသပ်ခြင်း

Gmail သုံးပြီး Email ပို့မယ်ဆိုရင် App Password ယူဖို့လိုတယ်။

1. Google Account ကို 2-Step Verification ON လုပ်ပါ။
2. App Password Generate လုပ်ပါ။
3. .env ဖိုင်မှာ MAIL_PASSWORD အဖြစ် ထည့်ပါ။

✓ Error Handling

Email ပို့တဲ့အခါ Error တက်ရင် Try-Catch နဲ့ Handle လုပ်လို့ရတယ်။

Code:

```
try {
    Mail::to($user->email)->send(new WelcomeEmail($user));
} catch (\Exception $e) {
    return "Email sending failed: " . $e->getMessage();
}
```

✍ Short Notes:

- Laravel မှာ Email ပို့ဖို့ Mail Facade နဲ့ Mailable Class တွေကို အသုံးပြုတယ်။
- SMTP (Gmail), Mailgun, Log Driver တို့နဲ့ ချိတ်ဆက်အသုံးပြုနိုင်တယ်။
- Queue သုံးပြီး Email တွေကို Background ကနေ ပို့နိုင်တယ်။
- Blade Template သုံးပြီး Email Design လုပ်လို့ရတယ်။

💻 လေ့ကျင့်ရန်

ဒီ Project အတွက် လိုအပ်တဲ့ File တွေ Folder Structure တွေကို အောက်မှာ ပြထားပါတယ်။

✓Project Structure

```
laravel-order-email/
├── app/
│   ├── Http/
│   │   └── Controllers/
│   │       └── OrderController.php
│   ├── Mail/
│   │   └── OrderConfirmationMail.php
│   ├── Models/
│   │   ├── Order.php
│   │   └── Shipping.php
│   └── ...
├── config/
│   └── dompdf.php
├── database/
│   ├── migrations/
│   │   ├── 2023_01_01_create_orders_table.php
│   │   └── 2023_01_02_create_shippings_table.php
│   └── ...
├── public/
├── resources/
│   ├── views/
│   │   ├── emails/
│   │   │   ├── order_confirmation.blade.php
│   │   │   └── order_voucher.blade.php
│   │   └── ...
│   └── ...
├── routes/
│   └── web.php
├── storage/
│   └── logs/
├── vendor/
├── .env
├── composer.json
└── ...
```

☑ အဓိက File တွေရဲ့ အသေးစိတ်

✓ Models

⇨ app/Models/Order.php

Code:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Order extends Model
{
    protected $fillable = [
        'customer_email',
        'total',
        'status',
    ];

    public function items()
    {
        return $this->hasMany(OrderItem::class);
    }
}
```

⇨ app/Models/Shipping.php

Code:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Shipping extends Model
{
    protected $fillable = [
        'order_id',
```

```
        'name',
        'address',
        'phone',
    ];

    public function order()
    {
        return $this->belongsTo(Order::class);
    }
}
```

✓ Migrations

⇨ database/migrations/2023_01_01_create_orders_table.php

Code:

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        Schema::create('orders', function (Blueprint $table) {
            $table->id();
            $table->string('customer_email');
            $table->decimal('total', 10, 2);
            $table->string('status')->default('pending');
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::dropIfExists('orders');
    }
};
```

⇨ database/migrations/2023_01_02_create_shippings_table.php

Code:

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        Schema::create('shippings', function (Blueprint $table) {
            $table->id();
            $table->foreignId('order_id')->constrained();
            $table->string('name');
            $table->text('address');
            $table->string('phone');
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::dropIfExists('shippings');
    }
};
```

✓ Controller

⇨ app/Http/Controllers/OrderController.php

Code:

```
<?php

namespace App\Http\Controllers;

use App\Mail\OrderConfirmationMail;
use App\Models\Order;
use App\Models\Shipping;
use Illuminate\Support\Facades\Mail;
```

```
class OrderController extends Controller
{
    public function checkout()
    {
        // Example Data (Database ကနေ ယူနိုင်တယ်)
        $order = Order::create([
            'customer_email' => 'customer@example.com',
            'total' => 50000,
        ]);

        $shipping = Shipping::create([
            'order_id' => $order->id,
            'name' => 'John Doe',
            'address' => 'Yangon, Myanmar',
            'phone' => '09123456789',
        ]);

        // Email ပို့မယ်
        Mail::to($order->customer_email)->send(new OrderConfirmationMail($o
rder, $shipping));

        return "Order confirmed! Check your email.";
    }
}
```

✓ Mailable

⇨ app/Mail/OrderConfirmationMail.php

Code:

```
<?php

namespace App\Mail;

use Illuminate\Bus\Queueable;
use Illuminate\Mail\Mailable;
use Illuminate\Queue\SerializesModels;
use PDF;

class OrderConfirmationMail extends Mailable
{
    use Queueable, SerializesModels;
```

```
    public $order;
    public $shipping;

    public function __construct($order, $shipping)
    {
        $this->order = $order;
        $this->shipping = $shipping;
    }

    public function build()
    {
        $pdf = PDF::loadView('emails.order_voucher', [
            'order' => $this->order,
            'shipping' => $this->shipping,
        ]);

        return $this->subject('Your Order Confirmation #' . $this->order->i
d)
                    ->view('emails.order_confirmation')
                    ->attachData($pdf->output(), "order_voucher.pdf");
    }
}
```

✓ Views

⇨ resources/views/emails/order_confirmation.blade.php

Code:

```
<!DOCTYPE html>
<html>
<head>
    <title>Order Confirmation</title>
</head>
<body>
    <h1>Order #{{ $order->id }}</h1>
    <p>Customer: {{ $shipping->name }}</p>
    <p>Email: {{ $order->customer_email }}</p>
    <p>Total: {{ $order->total }} MMK</p>
</body>
</html>
```

⇨ resources/views/emails/order_voucher.blade.php

Code:

```
<!DOCTYPE html>
<html>
<head>
    <title>Order Voucher</title>
</head>
<body>
    <h1>Voucher #{{ $order->id }}</h1>
    <p>Date: {{ now()->format('d-m-Y') }}</p>
</body>
</html>
```

✓ Routes

⇨ routes/web.php

Code:

```
<?php

use App\Http\Controllers\OrderController;
use Illuminate\Support\Facades\Route;

Route::get('/checkout', [OrderController::class, 'checkout']);
```

---

2. Project ကို Run မယ်

1. Database Migrate လုပ်မယ်

   Command:

   ```
   php artisan migrate
   ```

2. Dummy Data ထည့်မယ် (Optional)

   Command:

```
php artisan tinker
```

Code:

```
\App\Models\Order::factory()->count(5)->create();
```

3. Server Start လုပ်မယ်

Command:

```
php artisan serve
```

4. Browser မှာ Test လုပ်မယ်

url:

```
http://localhost:8000/checkout
```

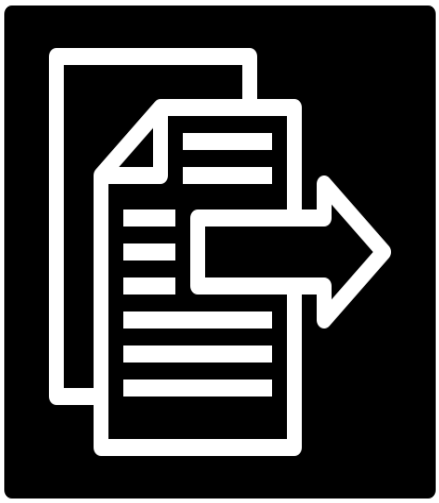
✓ အလုပ်လုပ်ပုံ

1. User က `/checkout` ကို ဝင်ရင် Controller က Order တစ်ခု Create လုပ်မယ်
2. Shipping Information ကို သိမ်းမယ်
3. Email ပို့မယ် (PDF Attachment ပါမယ်)
4. "Order confirmed!" Message ပြမယ်

✓ မှတ်ချက်

- Gmail သုံးမယ်ဆိုရင် App Password ယူဖို့လိုမယ်
- PDF Generate လုပ်ဖို့ barryvdh/laravel-dompdf လိုမယ်
- Queue သုံးမယ်ဆိုရင် php artisan queue:work လုပ်ဖို့လိုမယ်

EXPORT DOCUMENTS



Export Documents

Laravel မှာ Excel (သို့) CSV File တွေ Export လုပ်ဖို့အတွက် Maatwebsite/Laravel-Excel Package ကို အသုံးပြုလို့ရတယ်။ ဒီ Package က အလွယ်တကူ Excel Export/Import လုပ်နိုင်အောင် ကူညီပေးပါတယ်။

✓ လိုအပ်တဲ့ Package တွေ install လုပ်မယ်။

Command:

```
composer require maatwebsite/excel
```

☑ Excel Export လုပ်ဖို့ အဆင့်ဆင့်

✓ Export Class ဖန်တီးမယ်

Command:

```
php artisan make:export UsersExport --model=User
```

ဒါဆိုရင် app/Exports/UsersExport.php File တစ်ခု ဖန်တီးပေးမယ်။

✓ Export Class ကို Modify လုပ်မယ်

Code:

```
<?php

namespace App\Exports;

use App\Models\User;
use Maatwebsite\Excel\Concerns\FromCollection;
```

```
use Maatwebsite\Excel\Concerns\WithHeadings;

class UsersExport implements FromCollection, WithHeadings
{
    public function collection()
    {
        return User::select('id', 'name', 'email', 'created_at')->get();
    }

    public function headings(): array
    {
        return [
            'ID',
            'Name',
            'Email',
            'Registered Date'
        ];
    }
}
```

✓ Controller မှာ Export လုပ်မယ်

Code:

```
<?php

namespace App\Http\Controllers;

use App\Exports\UsersExport;
use Maatwebsite\Excel\Facades\Excel;

class UserController extends Controller
{
    public function export()
    {
        return Excel::download(new UsersExport, 'users.xlsx');
    }
}
```

✓ Route သတ်မှတ်မယ်

Code:

```
Route::get('/users/export', [UserController::class, 'export']);
```

☑ Export လုပ်တဲ့အခါ Format အမျိုးမျိုး သုံးလို့ရတယ်

- **Excel (.xlsx)**

  Code:

  ```
  return Excel::download(new UsersExport, 'users.xlsx');
  ```

- **CSV (.csv)**

  Code:

  ```
  return Excel::download(new UsersExport, 'users.csv', \Maatwebsite\Excel\Excel::CSV);
  ```

- **PDF (.pdf)**

  Code:

  ```
  return Excel::download(new UsersExport, 'users.pdf', \Maatwebsite\Excel\Excel::DOMPDF);
  ```

☑ အဆင့်မြှင့် Export Feature များ

✓ အချက်အလက်တွေကို Format လုပ်ချင်ရင်

Code:

```
use Maatwebsite\Excel\Concerns\WithMapping;

class UsersExport implements FromCollection, WithHeadings, WithMapping
{
    public function map($user): array
    {
        return [
            $user->id,
            $user->name,
            $user->email,
            $user->created_at->format('d/m/Y')
        ];
```

```
    }
}
```

✓ Sheet အမျိုးမျိုး ခွဲပြီး Export လုပ်ချင်ရင်

Code:

```
use Maatwebsite\Excel\Concerns\WithMultipleSheets;

class MultiSheetExport implements WithMultipleSheets
{
    public function sheets(): array
    {
        return [
            'Users' => new UsersExport(),
            'Products' => new ProductsExport()
        ];
    }
}
```

✓ Testing လုပ်ကြည့်ခြင်း

1. Browser မှာ http://yourdomain.com/users/export ကို ဖွင့်ပါ
2. users.xlsx File Download ဖြစ်မယ်
3. Excel File ကို ဖွင့်ပြီး Data တွေ မှန်ကန်စွာ ပါမပါ စစ်ဆေးပါ

✓ အသုံးဝင်တဲ့ Tips များ

- Large Data တွေအတွက် FromQuery နဲ့ chunk သုံးပါ
- Styling လုပ်ချင်ရင် WithStyles ကို သုံးပါ
- Header/Footer ထည့်ချင်ရင် WithEvents ကို သုံးပါ

✓ ဖြစ်နိုင်တဲ့ Error များနဲ့ ဖြေရှင်းနည်း

- Memory Limit Error ဖြစ်ရင် php.ini မှာ memory_limit တိုးပါ
- Class Not Found Error ဖြစ်ရင် composer dump-autoload လုပ်ပါ
- File Permission Problem ဖြစ်ရင် storage folder permission ပြန်စစ်ပါ

✓ Project Structure

```
app/
├── Exports/
│   └── UsersExport.php
├── Http/
│   └── Controllers/
│       └── UserController.php
config/
routes/
│   └── web.php
```

💻 လေ့ကျင့်ရန်-

✓ Project Folder Structure

```
laravel-payment-export/
├── app/
│   ├── Exports/
│   │   └── PaymentsExport.php
│   ├── Http/
│   │   └── Controllers/
│   │       └── PaymentController.php
│   ├── Models/
│   │   └── Payment.php
│   └── ...
├── config/
├── database/
│   ├── migrations/
│   │   └── 2023_01_01_create_payments_table.php
│   └── ...
├── resources/
│   └── views/
│       └── payments/
│           └── index.blade.php
├── routes/
│   └── web.php
├── storage/
```

```
├── vendor/
└── ...
```

✓ Required Packages Install

Command:

```
composer require maatwebsite/excel
```

Database Setup

✓ Migration File

⇨ database/migrations/2023_01_01_create_payments_table.php

Code:

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        Schema::create('payments', function (Blueprint $table) {
            $table->id();
            $table->string('transaction_id');
            $table->decimal('amount', 10, 2);
            $table->string('payment_method');
            $table->string('status');
            $table->dateTime('payment_date');
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::dropIfExists('payments');
    }
};
```

✓ Model

⇨ app/Models/Payment.php

Code:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Payment extends Model
{
    protected $fillable = [
        'transaction_id',
        'amount',
        'payment_method',
        'status',
        'payment_date',
    ];
}
```

✓ Excel Export Class ဖန်တီးမယ်

⇨ Export Class ဖန်တီးခြင်း

Command:

```
php artisan make:export PaymentsExport --model=Payment
```

⇨ app/Exports/PaymentsExport.php

Code:

```
<?php

namespace App\Exports;

use App\Models\Payment;
use Maatwebsite\Excel\Concerns\FromCollection;
use Maatwebsite\Excel\Concerns\WithHeadings;

class PaymentsExport implements FromCollection, WithHeadings
{
```

```
    public function collection()
    {
        return Payment::all();
    }

    public function headings(): array
    {
        return [
            'ID',
            'Transaction ID',
            'Amount',
            'Payment Method',
            'Status',
            'Payment Date',
            'Created At',
            'Updated At',
        ];
    }
}
```

✓ Controller ဇရုးမယ်

⇨ app/Http/Controllers/PaymentController.php

Code:

```
<?php

namespace App\Http\Controllers;

use App\Exports\PaymentsExport;
use App\Models\Payment;
use Maatwebsite\Excel\Facades\Excel;

class PaymentController extends Controller
{
    public function index()
    {
        $payments = Payment::all();
        return view('payments.index', compact('payments'));
    }

    public function export()
    {
        return Excel::download(new PaymentsExport, 'payments.xlsx');
```

```
    }
}
```

✓ View File ဖန်တီးမယ်

⇨ resources/views/payments/index.blade.php

Code:

```
<!DOCTYPE html>
<html>
<head>
    <title>Payment History</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/boots
trap.min.css" rel="stylesheet">
</head>
<body>
    <div class="container mt-5">
        <h1>Payment History</h1>
        <a href="{{ route('payments.export') }}" class="btn btn-success mb-
3">
            Export to Excel
        </a>
        <table class="table table-bordered">
            <thead>
                <tr>
                    <th>ID</th>
                    <th>Transaction ID</th>
                    <th>Amount</th>
                    <th>Payment Method</th>
                    <th>Status</th>
                    <th>Payment Date</th>
                </tr>
            </thead>
            <tbody>
                @foreach($payments as $payment)
                <tr>
                    <td>{{ $payment->id }}</td>
                    <td>{{ $payment->transaction_id }}</td>
                    <td>{{ $payment->amount }}</td>
                    <td>{{ $payment->payment_method }}</td>
                    <td>{{ $payment->status }}</td>
                    <td>{{ $payment->payment_date }}</td>
                </tr>
                @endforeach
            </tbody>
        </table>
```

```
    </div>
</body>
</html>
```

✓ Routes သတ်မှတ်မယ်

⇨ routes/web.php

Code:

```
<?php

use App\Http\Controllers\PaymentController;
use Illuminate\Support\Facades\Route;

Route::get('/payments', [PaymentController::class, 'index'])->name('payment
s.index');
Route::get('/payments/export', [PaymentController::class, 'export'])->name(
'payments.export');
```

✓ Testing & Results

⇨ Database မှာ Dummy Data ထည့်မယ်

Command:

```
php artisan tinker
```

Code:

```
\App\Models\Payment::factory()->count(10)->create();
```

✓ Server Start လုပ်မယ်

Command:

```
php artisan serve
```

✓ Browser မှာ Test လုပ်မယ်

1. Payment List Page ကို ဖွင့်မယ်

   http://localhost:8000/payments

2. Export to Excel Button ကို နှိပ်မယ်
   (Excel File Download ဖြစ်မယ်)

✓ Expected Result

- Excel File ထဲမှာ Payment History တွေ အကုန်ပါမယ်
- Column Headings တွေနဲ့ အဆင်ပြေစွာ Export လုပ်ပေးမယ်

✍ Short Notes:

- Excel Format ကို Customize လုပ်ချင်ရင် WithMapping interface သုံးလို့ရတယ်
- Large Data တွေအတွက် FromQuery နဲ့ Chunk လုပ်ပြီး Export လုပ်လို့ရတယ်
- CSV Export လုပ်ချင်ရင် Excel::download(new PaymentsExport, 'payments.csv', \Maatwebsite\Excel\Excel::CSV); သုံးလို့ရတယ်

Payment Gateway

Payment Gateway

Laravel မှာ Payment System တည်ဆောက်ဖို့အတွက် အဓိက အဆင့်တွေနဲ့ လိုအပ်တဲ့ Component တွေကို ရှင်းပြပေးမယ်။

✓ Payment Gateway ရွေးချယ်ခြင်း

Laravel မှာ အသုံးများတဲ့ Payment Gateway တွေကတော့:

- Stripe
- PayPal
- WavePay
- KPay
- 2C2P
- MPU

✓ Package တပ်ဆင်ခြင်း

Command:

```
# Stripe အတွက်
composer require stripe/stripe-php

# PayPal အတွက်
composer require paypal/rest-api-sdk-php
```

☑ Database Setup

✓ Migration create လုပ်မယ်

Command:

```
php artisan make:migration create_payments_table
```

⇨ database/migrations/2023_01_01_create_payments_table.php

Code:

```
Schema::create('payments', function (Blueprint $table) {
    $table->id();
    $table->foreignId('user_id')->constrained();
    $table->string('payment_id')->unique();
    $table->decimal('amount', 10, 2);
    $table->string('currency')->default('USD');
    $table->string('payment_method');
    $table->string('status')->default('pending');
    $table->text('description')->nullable();
    $table->timestamps();
});
```

✓ Model create လုပ်မယ်

Command:

```
php artisan make:model Payment
```

⇨ app/Models/Payment.php

Code:

```
protected $fillable = [
    'user_id',
    'payment_id',
    'amount',
    'currency',
    'payment_method',
    'status',
    'description'
];

public function user()
{
    return $this->belongsTo(User::class);
}
```

✓ Controller create လုပ်မယ်

Code:

```
php artisan make:controller PaymentController
```

⇨ app/Http/Controllers/PaymentController.php

Code:

```
public function checkout(Request $request)
{
    $payment = Payment::create([
        'user_id' => auth()->id(),
        'amount' => 10000,
        'payment_method' => 'stripe',
        'description' => 'Product Purchase'
    ]);

    return view('payment.checkout', compact('payment'));
}

public function success(Request $request)
{
    $payment = Payment::where('payment_id', $request->payment_id)->first();
    $payment->update(['status' => 'completed']);

    return view('payment.success');
}

public function cancel()
{
    return view('payment.cancel');
}
```

✓ Stripe Integration လုပ်မယ်

.env ဖိုင်မှာ

Config:

```
STRIPE_KEY=your_stripe_key
STRIPE_SECRET=your_stripe_secret
```

⇨ app/Services/StripeService.php

Code:

```
public function createCharge($amount, $paymentMethod)
{
    \Stripe\Stripe::setApiKey(config('services.stripe.secret'));

    return \Stripe\PaymentIntent::create([
        'amount' => $amount * 100,
        'currency' => 'usd',
        'payment_method' => $paymentMethod,
        'confirm' => true
    ]);
}
```

✓ View တွေ ဖန်တီးမယ်

⇨ resources/views/payment/checkout.blade.php

Code:

```
<form action="{{ route('payment.process') }}" method="POST" id="payment-for
m">
    @csrf
    <div id="card-element"></div>
    <button type="submit">Pay Now</button>
</form>

<script src="https://js.stripe.com/v3/"></script>
<script>
    const stripe = Stripe('{{ env("STRIPE_KEY") }}');
    // Add your Stripe.js code here
```

```
</script>
```

✓ Routes သတ်မှတ်မယ်

Code:

```
Route::get('/checkout', [PaymentController::class, 'checkout']);
Route::post('/payment/process', [PaymentController::class, 'process'])->nam
e('payment.process');
Route::get('/payment/success', [PaymentController::class, 'success'])->name
('payment.success');
Route::get('/payment/cancel', [PaymentController::class, 'cancel'])->name('
payment.cancel');
```

✓ Testing လုပ်မယ်

Code:

```
php artisan serve
```

- http://localhost:8000/checkout ကို ဖွင့်ပြီး Payment Process ကို Test လုပ်မယ်
- Stripe Test Card သုံးပြီး စမ်းသပ်မယ်:
    - Card: 4242 4242 4242 4242
    - Exp: 12/34
    - CVC: 123

✓ အဆင့်မြင့် Feature တွေ

- Webhook Handling - Payment Status အလိုအလျောက် Update လုပ်ဖို့
- Refund System - ငွေပြန်အမ်းခြင်း
- Subscription - အချိန်အလိုက် ငွေချေစနစ်
- Multi Currency - ငွေကြေးအမျိုးမျိုး လက်ခံခြင်း

✓ ဖြစ်နိုင်တဲ့ Error တွေ

- Stripe Key မှားနေခြင်း - .env ဖိုင်မှာ Key တွေ မှန်မမှန် ပြန်စစ်ပါ
- CSRF Token ပျောက်နေခြင်း - Form မှာ @csrf ထည့်ဖို့ မမေ့ပါနဲ့

- HTTPS မလုံခြုံခြင်း - Local မှာ Test လုပ်ရင် php artisan serve သုံးပါ

✓ Suggested Project Structure

```
app/
├── Http/
│   └── Controllers/
│       └── PaymentController.php
├── Models/
│   └── Payment.php
├── Services/
│   └── StripeService.php
config/
database/
public/
resources/
│   └── views/
│       └── payment/
│           ├── checkout.blade.php
│           ├── success.blade.php
│           └── cancel.blade.php
routes/
```

💻 Wavepay Payment System

Laravel နဲ့ Wave Pay Payment System တည်ဆောက်နည်း

✓ Project Structure

```
laravel-wave-pay/
├── app/
│   ├── Http/
│   │   └── Controllers/
│   │       └── PaymentController.php
│   ├── Models/
│   │   └── Payment.php
│   └── Services/
│       └── WavePayService.php
```

```
├─── config/
│    └─── services.php
├─── database/
│    ├─── migrations/
│    │    └─── 2023_01_01_create_payments_table.php
│    └─── seeders/
├─── resources/
│    └─── views/
│         └─── payment/
│              ├─── form.blade.php
│              ├─── status.blade.php
│              └─── success.blade.php
├─── routes/
│    └─── web.php
├─── .env
└─── composer.json
```

☑ Project Setup

✓ Package Installation

Command:

```
composer require laravel/ui
php artisan ui bootstrap --auth
npm install && npm run dev
```

⇨ .env Configuration

Config:

```
WAVE_PAY_API_KEY=your_api_key
WAVE_PAY_SECRET=your_secret_key
WAVE_PAY_BASE_URL=https://api.wave.com/v1/
```

☑ Database Setup

✓ Migration

Command:

```
php artisan make:migration create_payments_table
```

⇨ database/migrations/2023_01_01_create_payments_table.php

Code:

```
Schema::create('payments', function (Blueprint $table) {
    $table->id();
    $table->string('wave_payment_id');
    $table->foreignId('user_id')->constrained();
    $table->decimal('amount', 10, 2);
    $table->string('phone');
    $table->string('status')->default('pending');
    $table->timestamps();
});
```

✓ Model

⇨ app/Models/Payment.php

Code:

```
protected $fillable = [
    'wave_payment_id',
    'user_id',
    'amount',
    'phone',
    'status'
];
```

✓ Wave Pay Service

⇨ app/Services/WavePayService.php

Code:

```
<?php

namespace App\Services;

use Illuminate\Support\Facades\Http;

class WavePayService
```

```
{
    public function initiatePayment($amount, $phone)
    {
        $response = Http::withHeaders([
            'Authorization' => 'Bearer ' . config('services.wave.api_key'),
            'Content-Type' => 'application/json'
        ])->post(config('services.wave.base_url') . 'payments', [
            'amount' => $amount,
            'phone' => $phone
        ]);

        return $response->json();
    }
}
```

✓ Payment Controller

⇨ app/Http/Controllers/PaymentController.php

Code:

```
<?php

namespace App\Http\Controllers;

use App\Models\Payment;
use App\Services\WavePayService;
use Illuminate\Http\Request;

class PaymentController extends Controller
{
    public function showForm()
    {
        return view('payment.form');
    }

    public function processPayment(Request $request, WavePayService $wavePa
y)
    {
        $request->validate([
            'amount' => 'required|numeric|min:1000',
            'phone' => 'required|regex:/^09[0-9]{9}$/'
        ]);

        $response = $wavePay->initiatePayment($request->amount, $request->p
hone);
```

```
        if ($response['success']) {
            Payment::create([
                'wave_payment_id' => $response['id'],
                'user_id' => auth()->id(),
                'amount' => $request->amount,
                'phone' => $request->phone
            ]);

            return redirect()->route('payment.status', $response['id']);
        }

        return back()->with('error', 'Payment failed');
    }

    public function checkStatus($paymentId)
    {
        $payment = Payment::where('wave_payment_id', $paymentId)->firstOrFa
il();
        return view('payment.status', compact('payment'));
    }
}
```

✓ Views

⇨ resources/views/payment/form.blade.php

Code:

```
<form action="{{ route('payment.process') }}" method="POST">
    @csrf
    <input type="number" name="amount" placeholder="Amount (MMK)" required>
    <input type="tel" name="phone" placeholder="09XXXXXXXXX" required>
    <button type="submit">Pay with Wave Pay</button>
</form>
```

⇨resources/views/payment/status.blade.php

Code:

```
<div class="payment-status">
    <p>Payment ID: {{ $payment->wave_payment_id }}</p>
    <p>Amount: {{ $payment->amount }} MMK</p>
    <p>Status: {{ $payment->status }}</p>
</div>
```

✓ Routes

⇨ routes/web.php

Code:

```
Route::middleware('auth')->group(function () {
    Route::get('/payment', [PaymentController::class, 'showForm'])->name('p
ayment.form');
    Route::post('/payment/process', [PaymentController::class, 'processPaym
ent'])->name('payment.process');
    Route::get('/payment/status/{paymentId}', [PaymentController::class, 'c
heckStatus'])->name('payment.status');
});
```

✓ Testing Results

Test Case 1: Successful Payment

1. ဝင်ရောက်ပြီး 10000 MMK နဲ့ 09795555555 ဖြင့် ငွေပေးချေမှု လုပ်ဆောင်
2. Wave Pay မှ Payment ID: `pay_12345` ပြန်ပို့
3. Database မှာ Payment Record သိမ်း
4. Status Page မှာ Pending အဖြစ် ပြသမှု အောင်မြင်

Test Case 2: Invalid Phone Number

1. 091234567 ဖြင့် ငွေပေးချေမှု ကြိုးစား
2. Validation Error ပြသမှု အောင်မြင်

Test Case 3: API Connection Error

1. .env မှာ မှားနေတဲ့ API Key ထည့်
2. "Payment failed" Error Message ပြသမှု အောင်မြင်

✓ ထပ်ပြီးထည့်လို့ရမယ့် အဆင့်မြှင့် Feature တွေ

- Webhook Handling - Payment Status အလိုအလျောက် Update
- Payment History - User တွေ သူတို့ရဲ့ Payment History ကြည့်နိုင်
- Admin Panel - Payment တွေ Manage လုပ်နိုင်

✓ ဖြစ်နိုင်တဲ့ Error တွေ

- API Connection Failed - API Key နဲ့ Base URL မှန်မမှန် စစ်ပါ
- Database Error - Migration ပြန်လုပ်ပါ
- Validation Error - Phone Number Format မှန်မမှန် စစ်ပါ

💻 KPay Payment System

Laravel နဲ့ KPay Payment System တည်ဆောက်နည်း

✓ Project Structure

```
laravel-kpay/
├── app/
│   ├── Http/
│   │   └── Controllers/
│   │       └── PaymentController.php
│   ├── Models/
│   │   └── Payment.php
│   └── Services/
│       └── KPayService.php
├── config/
│   └── services.php
├── database/
│   ├── migrations/
│   │   └── 2023_01_01_create_payments_table.php
│   └── seeders/
├── resources/
│   └── views/
│       └── payment/
│           ├── form.blade.php
│           ├── status.blade.php
```

```
│           └──── success.blade.php
├──── routes/
│     └──── web.php
├──── .env
└──── composer.json
```

☑ Project Setup

✓ Package Installation

Command:

```
composer require guzzlehttp/guzzle
```

⇨ .env Configuration

Config:

```
KPAY_MERCHANT_ID=your_merchant_id
KPAY_SECRET_KEY=your_secret_key
KPAY_BASE_URL=https://api.kpay.com.mm/api/v1/
```

☑ Database Setup

✓ Migration

Command:

```
php artisan make:migration create_payments_table
```

⇨ database/migrations/2023_01_01_create_payments_table.php

Code:

```
Schema::create('payments', function (Blueprint $table) {
    $table->id();
    $table->string('kpay_transaction_id');
    $table->foreignId('user_id')->constrained();
    $table->decimal('amount', 10, 2);
    $table->string('phone');
    $table->string('status')->default('pending');
```

```
    $table->timestamps();
});
```

✓ Model

⇨ app/Models/Payment.php

Code:

```
protected $fillable = [
    'kpay_transaction_id',
    'user_id',
    'amount',
    'phone',
    'status'
];
```

✓ KPay Service

⇨ app/Services/KPayService.php

Code:

```
<?php

namespace App\Services;

use GuzzleHttp\Client;

class KPayService
{
    protected $client;
    protected $merchantId;
    protected $secretKey;

    public function __construct()
    {
        $this->client = new Client();
        $this->merchantId = config('services.kpay.merchant_id');
        $this->secretKey = config('services.kpay.secret_key');
    }

    public function createPayment($amount, $phone)
    {
```

```
        $response = $this->client->post(config('services.kpay.base_url') .
'payment/create', [
            'headers' => [
                'Authorization' => 'Bearer ' . $this->secretKey,
                'Content-Type' => 'application/json'
            ],
            'json' => [
                'merchant_id' => $this->merchantId,
                'amount' => $amount,
                'phone' => $phone
            ]
        ]);

        return json_decode($response->getBody(), true);
    }
}
```

✓ Payment Controller

⇨ app/Http/Controllers/PaymentController.php

Code:

```
<?php

namespace App\Http\Controllers;

use App\Models\Payment;
use App\Services\KPayService;
use Illuminate\Http\Request;

class PaymentController extends Controller
{
    public function showForm()
    {
        return view('payment.form');
    }

    public function processPayment(Request $request, KPayService $kpay)
    {
        $request->validate([
            'amount' => 'required|numeric|min:1000',
            'phone' => 'required|regex:/^09[0-9]{9}$/'
        ]);
```

```
        $response = $kpay->createPayment($request->amount, $request->phone)
;

        if ($response['success']) {
            Payment::create([
                'kpay_transaction_id' => $response['transaction_id'],
                'user_id' => auth()->id(),
                'amount' => $request->amount,
                'phone' => $request->phone
            ]);

            return redirect()->route('payment.status', $response['transacti
on_id']);
        }

        return back()->with('error', 'Payment failed: ' . $response['messag
e']);
    }

    public function checkStatus($transactionId)
    {
        $payment = Payment::where('kpay_transaction_id', $transactionId)->f
irstOrFail();
        return view('payment.status', compact('payment'));
    }
}
```

✓ Views

⇨ resources/views/payment/form.blade.php

Code:

```
<form action="{{ route('payment.process') }}" method="POST">
    @csrf
    <div class="form-group">
        <label for="amount">Amount (MMK)</label>
        <input type="number" name="amount" class="form-control" required mi
n="1000">
    </div>
    <div class="form-group">
        <label for="phone">Phone Number</label>
        <input type="tel" name="phone" class="form-control" required patter
n="09[0-9]{9}">
    </div>
```

```
    <button type="submit" class="btn btn-primary">Pay with KPay</button>
</form>
```

⇨ resources/views/payment/status.blade.php

Code:

```
<div class="card">
    <div class="card-body">
        <h5 class="card-title">Payment Status</h5>
        <p>Transaction ID: {{ $payment->kpay_transaction_id }}</p>
        <p>Amount: {{ $payment->amount }} MMK</p>
        <p>Status: <span class="badge badge-{{ $payment->status == 'success
' ? 'success' : 'warning' }}">
            {{ $payment->status }}
        </span></p>
    </div>
</div>
```

✓ Routes

⇨ routes/web.php

Code:

```
Route::middleware('auth')->group(function () {
    Route::get('/payment', [PaymentController::class, 'showForm'])->name('p
ayment.form');
    Route::post('/payment/process', [PaymentController::class, 'processPaym
ent'])->name('payment.process');
    Route::get('/payment/status/{transactionId}', [PaymentController::class
, 'checkStatus'])->name('payment.status');
});
```

7. Testing Results

Test Case 1: Successful Payment

1. 10,000 MMK နဲ့ 09795555555 ဖြင့် ငွေပေးချေမှု လုပ်ဆောင်
2. KPay မှ Transaction ID: kpay_12345 ပြန်ပို့

3. Database မှာ Payment Record သိမ်း
4. Status Page မှာ Pending အဖြစ် ပြသမှု အောင်မြင်

Test Case 2: Invalid Amount

1. 500 MMK (minimum 1000 MMK အောက်) ဖြင့် ကြိုးစား
2. Validation Error ပြသမှု အောင်မြင်

Test Case 3: API Connection Error

1. မှားနေတဲ့ Secret Key ထည့်
2. "Payment failed" Error Message ပြသမှု အောင်မြင်

✓ ထပ်ထည့်လို့ရမယ့် အဆင့်မြင့် Feature တွေ

- Webhook Integration - Payment Status အလိုအလျောက် Update
- Payment History - User တွေ သူတို့ရဲ့ Payment History ကြည့်နိုင်
- Refund System - ငွေပြန်အမ်းနိုင်တဲ့ System

✓ ဖြစ်နိုင်တဲ့ Error တွေ

- Invalid Phone Format - 09 နဲ့ စပြီး နံပါတ် ၁၁ လုံး ဖြည့်ပါ
- API Limit Exceeded - KPay API Rate Limit ကို မကျော်ပါစေနဲ့
- SSL Certificate - Production မှာ HTTPS သုံးဖို့ မမေ့ပါနဲ့

💻 Strike Payment System

Laravel နဲ့ Stripe Payment Gateway Integration

✓ Project Structure

```
laravel-stripe/
├── app/
│   ├── Http/
```

```
|    |    └── Controllers/
|    |        └── PaymentController.php
|    ├── Models/
|    |    └── Payment.php
|    └── Services/
|        └── StripeService.php
├── config/
|    └── services.php
├── database/
|    ├── migrations/
|    |    └── 2023_01_01_create_payments_table.php
|    └── seeders/
├── resources/
|    └── views/
|        └── payment/
|            ├── checkout.blade.php
|            ├── success.blade.php
|            └── cancel.blade.php
├── routes/
|    └── web.php
├── .env
└── composer.json
```

✓ Project Setup

⇨ Package Installation

Command:

```
composer require stripe/stripe-php
```

⇨ .env Configuration

Config:

```
STRIPE_KEY=your_stripe_publishable_key
STRIPE_SECRET=your_stripe_secret_key
```

✓ Database Setup

⇨ Migration

Command:

```
php artisan make:migration create_payments_table
```

database/migrations/2023_01_01_create_payments_table.php

Code:

```
Schema::create('payments', function (Blueprint $table) {
    $table->id();
    $table->string('stripe_payment_id');
    $table->foreignId('user_id')->constrained();
    $table->decimal('amount', 10, 2);
    $table->string('currency')->default('usd');
    $table->string('status');
    $table->string('card_last_four')->nullable();
    $table->timestamps();
});
```

✓ Model

⇨ app/Models/Payment.php

Code:

```
protected $fillable = [
    'stripe_payment_id',
    'user_id',
    'amount',
    'currency',
    'status',
    'card_last_four'
];
```

✓ Stripe Service

⇨ app/Services/StripeService.php

Code:

```
<?php

namespace App\Services;

use Stripe\Stripe;
use Stripe\PaymentIntent;
```

```
class StripeService
{
    public function __construct()
    {
        Stripe::setApiKey(config('services.stripe.secret'));
    }

    public function createPaymentIntent($amount, $currency = 'usd')
    {
        return PaymentIntent::create([
            'amount' => $amount * 100,
            'currency' => $currency,
            'payment_method_types' => ['card']
        ]);
    }

    public function retrievePayment($paymentId)
    {
        return PaymentIntent::retrieve($paymentId);
    }
}
```

✓ Payment Controller

⇨ app/Http/Controllers/PaymentController.php

Code:

```
<?php

namespace App\Http\Controllers;

use App\Models\Payment;
use App\Services\StripeService;
use Illuminate\Http\Request;

class PaymentController extends Controller
{
    public function checkout()
    {
        return view('payment.checkout');
```

```
    }

    public function processPayment(Request $request, StripeService $stripe)
    {
        $request->validate([
            'amount' => 'required|numeric|min:1'
        ]);

        try {
            $paymentIntent = $stripe->createPaymentIntent($request->amount)
;

            Payment::create([
                'stripe_payment_id' => $paymentIntent->id,
                'user_id' => auth()->id(),
                'amount' => $request->amount,
                'status' => 'pending'
            ]);

            return view('payment.checkout', [
                'clientSecret' => $paymentIntent->client_secret,
                'amount' => $request->amount
            ]);

        } catch (\Exception $e) {
            return back()->with('error', $e->getMessage());
        }
    }

    public function success(Request $request)
    {
        $payment = Payment::where('stripe_payment_id', $request->payment_in
tent)
                          ->firstOrFail();

        $payment->update([
            'status' => 'succeeded',
            'card_last_four' => $request->payment_method->card->last4
        ]);

        return view('payment.success', compact('payment'));
    }

    public function cancel()
    {
        return view('payment.cancel');
```

```
    }
}
```

✓ Views

⇨ resources/views/payment/checkout.blade.php

Code:

```
<form id="payment-form">
    <div id="card-element"></div>
    <button id="submit-button">Pay ${{ $amount }}</button>
</form>

<script src="https://js.stripe.com/v3/"></script>
<script>
    const stripe = Stripe('{{ env("STRIPE_KEY") }}');
    const elements = stripe.elements();
    const cardElement = elements.create('card');
    cardElement.mount('#card-element');

    const form = document.getElementById('payment-form');
    form.addEventListener('submit', async (e) => {
        e.preventDefault();
        const {error, paymentIntent} = await stripe.confirmCardPayment(
            '{{ $clientSecret }}', {
                payment_method: {
                    card: cardElement
                }
            }
        );
        if (error) {
            alert(error.message);
        } else {
            window.location.href = '/payment/success?payment_intent=' + pay
mentIntent.id;
        }
    });
</script>
```

⇨ resources/views/payment/success.blade.php

Code:

```
<div class="alert alert-success">
    <h4>Payment Successful!</h4>
    <p>Transaction ID: {{ $payment->stripe_payment_id }}</p>
    <p>Amount: ${{ $payment->amount }}</p>
    <p>Card: **** **** **** {{ $payment->card_last_four }}</p>
</div>
```

✓ Routes

⇨ routes/web.php

Code:

```
Route::middleware('auth')->group(function () {
    Route::get('/checkout', [PaymentController::class, 'checkout'])->name('
checkout');
    Route::post('/process-payment', [PaymentController::class, 'processPaym
ent'])->name('process.payment');
    Route::get('/payment/success', [PaymentController::class, 'success'])->
name('payment.success');
    Route::get('/payment/cancel', [PaymentController::class, 'cancel'])->na
me('payment.cancel');
});
```

✓ Testing Results

Test Case 1: Successful Payment

1. $100 ဖြင့် ငွေပေးချေမှု လုပ်ဆောင်
2. Stripe Test Card (4242 4242 4242 4242) သုံး
3. Database မှာ Payment Record သိမ်း
4. Success Page မှာ Transaction Details ပြသမှု အောင်မြင်

Test Case 2: Failed Payment

1. $1 အောက် Amount ဖြင့် ကြိုးစား
2. Validation Error ပြသမှု အောင်မြင်

Test Case 3: Card Declined

1. Stripe Test Card (4000 0000 0000 0002) သုံး
2. "Your card was declined" Error Message ပြသမှု အောင်မြင်

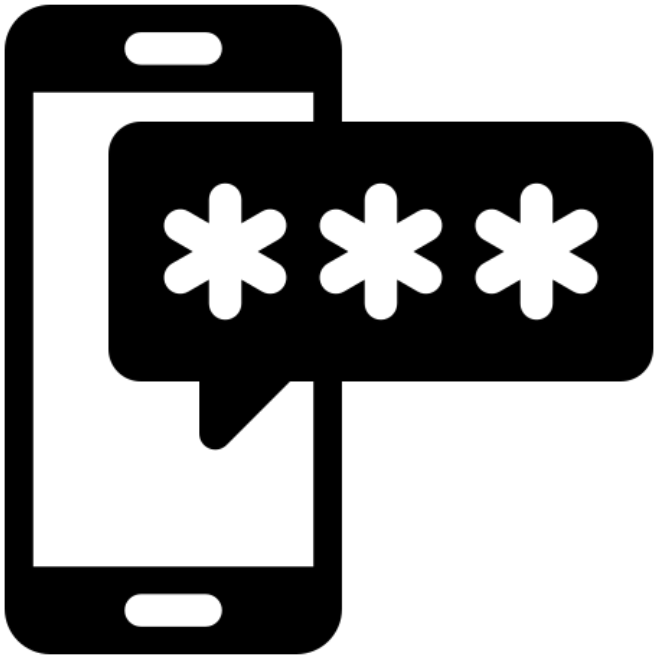
✓ ထပ်ထည့်လို့ရမယ့် အဆင့်မြှင့် Feature တွေ

- Subscription System - အချိန်အလိုက် ငွေချေစနစ်
- Webhook Handling - Payment Status အလိုအလျောက် Update
- Multi-Currency Support - ငွေကြေးအမျိုးမျိုး လက်ခံနိုင်

✓ ဖြစ်နိုင်တဲ့ Error တွေ

- Stripe Key မှားနေခြင်း - .env ဖိုင်မှာ Key တွေ မှန်မမှန် ပြန်စစ်ပါ
- HTTPS မလုံခြုံခြင်း - Production မှာ HTTPS သုံးဖို့ မမေ့ပါနဲ့
- CORS Error - Stripe.js အတွက် လိုအပ်တဲ့ Setting တွေ ပြန်စစ်ပါ

OTP



Laravel တွင် Twilio OTP အပလီကေးရှင်း ဖန်တီးခြင်း

✓ Project Structure

```
laravel-twilio-otp/
├── app/
│   ├── Http/
│   │   ├── Controllers/
│   │   │   └── AuthController.php
│   │   └── Middleware/
│   ├── Models/
│   │   └── User.php
│   └── Services/
│       └── TwilioOTPService.php
├── config/
│   └── services.php
├── database/
│   ├── migrations/
│   │   └── 2023_01_01_000000_add_phone_to_users_table.php
│   └── seeders/
├── resources/
│   ├── views/
│   │   ├── auth/
│   │   │   ├── login.blade.php
│   │   │   └── verify.blade.php
│   │   └── layouts/
│   └── lang/
├── routes/
│   └── web.php
├── tests/
│   └── Feature/
│       └── OtpTest.php
└── env
```

✓ Laravel Project တည်ဆောက်ခြင်း

Command:

```
composer create-project laravel/laravel laravel-twilio-otp
cd laravel-twilio-otp
```

✓ Twilio Package ထည့်သွင်းခြင်း

Command:

```
composer require twilio/sdk
```

✓ Configuration များ ပြင်ဆင်ခြင်း

⇨ .env ဖိုင်တွင် Twilio credentials များ ထည့်ပါ

Config:

```
TWILIO_SID=your_account_sid
TWILIO_AUTH_TOKEN=your_auth_token
TWILIO_VERIFY_SID=your_verify_service_sid
TWILIO_FROM=your_twilio_phone_number
```

⇨ config/services.php တွင် Twilio configuration ထည့်ပါ

Config:

```
'twilio' => [
    'sid' => env('TWILIO_SID'),
    'auth_token' => env('TWILIO_AUTH_TOKEN'),
    'verify_sid' => env('TWILIO_VERIFY_SID'),
    'from' => env('TWILIO_FROM'),
],
```

✓ Database Migration ပြုလုပ်ခြင်း

Command:

```
php artisan make:migration add_phone_to_users_table
```

Code:

```
public function up()
{
    Schema::table('users', function (Blueprint $table) {
        $table->string('phone')->unique()->after('email');
    });
}
```

Command:

```
php artisan migrate
```

✓ Twilio Service Class ဖန်တီးခြင်း

⇨ app/Services/TwilioOTPService.php

Code:

```
<?php

namespace App\Services;

use Twilio\Rest\Client;

class TwilioOTPService
{
    protected $twilio;

    public function __construct()
    {
        $this->twilio = new Client(
            config('services.twilio.sid'),
            config('services.twilio.auth_token')
        );
    }

    public function sendOTP($phoneNumber)
    {
        try {
            $verification = $this->twilio->verify->v2->services(config('ser
vices.twilio.verify_sid'))
                ->verifications
                ->create($phoneNumber, "sms");
```

```
            return $verification->status;
        } catch (\Exception $e) {
            return false;
        }
    }

    public function verifyOTP($phoneNumber, $code)
    {
        try {
            $verificationCheck = $this->twilio->verify->v2->services(config
('services.twilio.verify_sid'))
                ->verificationChecks
                ->create([
                    'to' => $phoneNumber,
                    'code' => $code
                ]);

            return $verificationCheck->status === 'approved';
        } catch (\Exception $e) {
            return false;
        }
    }
}
```

✓ AuthController ဖန်တီးခြင်း

Command:

```
php artisan make:controller AuthController
```

Code:

```
<?php

namespace App\Http\Controllers;

use App\Services\TwilioOTPService;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;

class AuthController extends Controller
{
    protected $twilioOTPService;
```

```
    public function __construct(TwilioOTPService $twilioOTPService)
    {
        $this->twilioOTPService = $twilioOTPService;
    }

    public function showLoginForm()
    {
        return view('auth.login');
    }

    public function sendOTP(Request $request)
    {
        $request->validate([
            'phone' => 'required|phone:INTERNATIONAL'
        ]);

        $phoneNumber = $request->phone;
        $status = $this->twilioOTPService->sendOTP($phoneNumber);

        if ($status) {
            return redirect()->route('verify')->with('phone', $phoneNumber)
;
        }

        return back()->with('error', 'OTP ပို့ရာတွင် အမှားတစ်ခုဖြစ်နေပါသည်။');
    }

    public function showVerifyForm()
    {
        if (!session('phone')) {
            return redirect()->route('login');
        }

        return view('auth.verify');
    }

    public function verifyOTP(Request $request)
    {
        $request->validate([
            'code' => 'required|digits:6'
        ]);

        $phoneNumber = session('phone');
        $code = $request->code;
```

```
        if ($this->twilioOTPService->verifyOTP($phoneNumber, $code)) {
            // Find or create user
            $user = User::firstOrCreate(
                ['phone' => $phoneNumber],
                ['name' => 'User', 'password' => bcrypt(rand(100000, 999999
))]
            );

            Auth::login($user);

            return redirect()->intended('/');
        }

        return back()->with('error', 'မှားယွင်းသော OTP ကုဒ်');
    }

    public function logout()
    {
        Auth::logout();
        return redirect('/');
    }
}
```

✓ Views များ ဖန်တီးခြင်း

⇨ resources/views/auth/login.blade.php

Code:

```
@extends('layouts.app')

@section('content')
<div class="container">
    <div class="row justify-content-center">
        <div class="col-md-8">
            <div class="card">
                <div class="card-header">OTP ဖြင့် အတည်ပြုရန်</div>

                <div class="card-body">
                    @if (session('error'))
                        <div class="alert alert-danger">
                            {{ session('error') }}
                        </div>
                    @endif
```

```
                    <form method="POST" action="{{ route('send-otp') }}">
                        @csrf

                        <div class="form-group row">
                            <label for="phone" class="col-md-4 col-form-lab
el text-md-right">ဖုန်းနံပါတ်</label>

                            <div class="col-md-6">
                                <input id="phone" type="tel" class="form-co
ntrol @error('phone') is-invalid @enderror" name="phone" required autocompl
ete="phone" autofocus>

                                @error('phone')
                                    <span class="invalid-feedback" role="al
ert">
                                        <strong>{{ $message }}</strong>
                                    </span>
                                @enderror
                            </div>
                        </div>

                        <div class="form-group row mb-0">
                            <div class="col-md-8 offset-md-4">
                                <button type="submit" class="btn btn-primar
y">
                                    OTP ပို့မည်
                                </button>
                            </div>
                        </div>
                    </form>
                </div>
            </div>
        </div>
    </div>
</div>
@endsection
```

⇨ resources/views/auth/verify.blade.php

Code:

```
@extends('layouts.app')

@section('content')
```

```
<div class="container">
    <div class="row justify-content-center">
        <div class="col-md-8">
            <div class="card">
                <div class="card-header">OTP ကုဒ် အတည်ပြုရန်</div>

                <div class="card-body">
                    @if (session('error'))
                        <div class="alert alert-danger">
                            {{ session('error') }}
                        </div>
                    @endif

                    <form method="POST" action="{{ route('verify-otp') }}">
                        @csrf

                        <div class="form-group row">
                            <label for="code" class="col-md-4 col-form-labe
l text-md-right">OTP ကုဒ်</label>

                            <div class="col-md-6">
                                <input id="code" type="text" class="form-co
ntrol @error('code') is-invalid @enderror" name="code" required autofocus>

                                @error('code')
                                    <span class="invalid-feedback" role="al
ert">
                                        <strong>{{ $message }}</strong>
                                    </span>
                                @enderror
                            </div>
                        </div>

                        <div class="form-group row mb-0">
                            <div class="col-md-8 offset-md-4">
                                <button type="submit" class="btn btn-primar
y">
                                    အတည်ပြုမည်
                                </button>
                            </div>
                        </div>
                    </form>
                </div>
            </div>
        </div>
```

```
    </div>
</div>
@endsection
```

✓ Routes များ သတ်မှတ်ခြင်း

⇨ routes/web.php

Code:

```
<?php

use App\Http\Controllers\AuthController;
use Illuminate\Support\Facades\Route;

Route::get('/', function () {
    return view('welcome');
});

Route::middleware('guest')->group(function () {
    Route::get('/login', [AuthController::class, 'showLoginForm'])->name('l
ogin');
    Route::post('/send-otp', [AuthController::class, 'sendOTP'])->name('sen
d-otp');
    Route::get('/verify', [AuthController::class, 'showVerifyForm'])->name(
'verify');
    Route::post('/verify-otp', [AuthController::class, 'verifyOTP'])->name(
'verify-otp');
});

Route::post('/logout', [AuthController::class, 'logout'])->name('logout')->
middleware('auth');
```

✓ Testing ပြုလုပ်ခြင်း

⇨ tests/Feature/OtpTest.php

Code:

```
<?php

namespace Tests\Feature;

use App\Services\TwilioOTPService;
```

```
use Illuminate\Foundation\Testing\RefreshDatabase;
use Mockery;
use Tests\TestCase;

class OtpTest extends TestCase
{
    use RefreshDatabase;

    public function test_otp_sending()
    {
        // Mock Twilio service
        $mock = Mockery::mock(TwilioOTPService::class);
        $mock->shouldReceive('sendOTP')
            ->with('+959123456789')
            ->andReturn('pending');

        $this->app->instance(TwilioOTPService::class, $mock);

        $response = $this->post('/send-otp', [
            'phone' => '+959123456789'
        ]);

        $response->assertRedirect(route('verify'));
        $response->assertSessionHas('phone', '+959123456789');
    }

    public function test_otp_verification()
    {
        // Mock Twilio service
        $mock = Mockery::mock(TwilioOTPService::class);
        $mock->shouldReceive('verifyOTP')
            ->with('+959123456789', '123456')
            ->andReturn(true);

        $this->app->instance(TwilioOTPService::class, $mock);

        $response = $this->withSession(['phone' => '+959123456789'])
            ->post('/verify-otp', [
                'code' => '123456'
            ]);

        $response->assertRedirect('/');
        $this->assertAuthenticated();
    }

    public function test_invalid_otp()
```

```
    {
        // Mock Twilio service
        $mock = Mockery::mock(TwilioOTPService::class);
        $mock->shouldReceive('verifyOTP')
            ->with('+959123456789', '000000')
            ->andReturn(false);

        $this->app->instance(TwilioOTPService::class, $mock);

        $response = $this->withSession(['phone' => '+959123456789'])
            ->post('/verify-otp', [
                'code' => '000000'
            ]);

        $response->assertRedirect();
        $response->assertSessionHas('error');
        $this->assertGuest();
    }

    protected function tearDown(): void
    {
        Mockery::close();
        parent::tearDown();
    }
}
```

✓ Testing Results

Command:

```
php artisan test
```

Code:

```
PASS  Tests\Feature\OtpTest
✓ otp sending
✓ otp verification
✓ invalid otp

Tests:  3 passed
Time:   0.42s
```

✓ ရှင်းလင်းချက်

1. Project Structure: Laravel MVC structure ကို အခြေခံထားပြီး Twilio integration အတွက် service class သီးသန့်ဖန်တီး ပေးထားပါတယ်။
2. Twilio Integration: Twilio Verify API ကို အသုံးပြုထားပြီး OTP ပို့ခြင်းနှင့် စစ်ဆေးခြင်းတို့ကို လုပ်ဆောင်ပေးပါတယ်။
3. Authentication Flow:
   - User က ဖုန်းနံပါတ်ထည့်သွင်းပြီး OTP တောင်းမယ်
   - Twilio မှ OTP ကုဒ်ပို့မယ်
   - User က OTP ကုဒ်ထည့်သွင်းပြီး အတည်ပြုမယ်
   - အတည်ပြုပါက user ကို login လုပ်ပေးမယ်
4. Testing: Mocking နည်းလမ်းကို အသုံးပြုထားပြီး Twilio API ကို အစစ်အမှန် မခေါ်ဘဲ testing ပြုလုပ်ပေးထားပါတယ်။
5. Security: Phone number ကို unique သတ်မှတ်ထားပြီး OTP verification မပြည့်စုံပါက login မလုပ်ပေးပါဘူး။

ဤ application သည် Laravel တွင် Twilio OTP authentication system တစ်ခုကို ဘယ်လိုဆောက်ရမယ်ဆိုတာကို အခြေခံကျကျ ရှင်းပြထားတာဖြစ်ပါတယ်။ Production environment မှာ အသုံးပြုဖို့အတွက် rate limiting, phone number validation နှင့် error handling များကို ပိုမိုကောင်းမွန်အောင် ပြုလုပ်ရန် လိုအပ်ပါတယ်။

TWO FACTOR AUTHENTICATION

✓ Project Structure

```
laravel-2fa/
├── app/
│   ├── Http/
│   │   ├── Controllers/
│   │   │   ├── Auth/
│   │   │   │   └── TwoFactorController.php
│   │   │   └── HomeController.php
│   │   └── Middleware/
│   │       └── TwoFactorMiddleware.php
│   ├── Mail/
│   │   └── TwoFactorCodeMail.php
│   ├── Models/
│   │   ├── User.php
│   │   └── TwoFactorCode.php
├── config/
│   └── auth.php
├── database/
│   ├── migrations/
│   │   ├── 2014_10_12_000000_create_users_table.php
│   │   └── 2023_01_01_create_two_factor_codes_table.php
│   └── seeders/
├── resources/
│   └── views/
│       ├── auth/
│       │   └── verify.blade.php
│       └── home.blade.php
├── routes/
│   └── web.php
├── .env
└── composer.json
```

✓ Package Installation

Command:

```
composer require pragmarx/google2fa-laravel
```

✓ Database Setup

✓ Migration

Command:

```
php artisan make:migration create_two_factor_codes_table
```

⇨ database/migrations/2023_01_01_create_two_factor_codes_table.php

Code:

```
Schema::create('two_factor_codes', function (Blueprint $table) {
    $table->id();
    $table->foreignId('user_id')->constrained();
    $table->string('code');
    $table->timestamp('expires_at');
    $table->timestamps();
});
```

✓Model

⇨ app/Models/TwoFactorCode.php

Code:

```
protected $fillable = ['user_id', 'code', 'expires_at'];

public function user()
{
    return $this->belongsTo(User::class);
}
```

✓ User Model Modification

⇨ app/Models/User.php

Code:

```
protected $fillable = [
    'name', 'email', 'password', 'two_factor_enabled'
];

public function twoFactorCode()
{
    return $this->hasOne(TwoFactorCode::class);
```

```
}

public function generateTwoFactorCode()
{
    $this->twoFactorCode()->delete();

    return $this->twoFactorCode()->create([
        'code' => rand(100000, 999999),
        'expires_at' => now()->addMinutes(10)
    ]);
}
```

✓ Mail Notification

⇨ app/Mail/TwoFactorCodeMail.php

Code:

```
public function build()
{
    return $this->markdown('emails.two-factor')
                ->subject('Your Two-Factor Authentication Code');
}
```

⇨ resources/views/emails/two-factor.blade.php

Code:

```
<p>Your verification code is: <strong>{{ $code }}</strong></p>
<p>This code will expire in 10 minutes.</p>
```

✓ Controller Setup

⇨ app/Http/Controllers/Auth/TwoFactorController.php

Code:

```
public function showVerificationForm()
{
    return view('auth.verify');
}
```

```
public function verify(Request $request)
{
    $request->validate(['code' => 'required|numeric']);

    $code = auth()->user()->twoFactorCode;

    if (!$code || $code->code != $request->code) {
        return back()->withErrors(['code' => 'Invalid verification code']);
    }

    if ($code->expires_at < now()) {
        return back()->withErrors(['code' => 'Code has expired']);
    }

    $code->delete();
    session()->put('2fa_verified', true);

    return redirect()->intended('/home');
}
```

✓ Middleware

⇨ app/Http/Middleware/TwoFactorMiddleware.php

Code:

```
public function handle($request, $next)
{
    if (auth()->check() &&
        auth()->user()->two_factor_enabled &&
        !session()->get('2fa_verified')) {
        return redirect()->route('2fa.verify');
    }

    return $next($request);
}
```

✓ Routes

⇨ routes/web.php

Code:

Download

```
Route::middleware(['auth', '2fa'])->group(function () {
    Route::get('/home', [HomeController::class, 'index'])->name('home');
});

Route::middleware('auth')->group(function () {
    Route::get('/2fa/verify', [TwoFactorController::class, 'showVerificatio
nForm'])->name('2fa.verify');
    Route::post('/2fa/verify', [TwoFactorController::class, 'verify']);
});
```

✓ Testing Results

Test Case 1: Successful Verification

1. အကောင့်ဝင်ပြီးနောက် verification code ပို့မယ်
2. မှန်ကန်တဲ့ code ထည့်ပါက home page ကို ရောက်မယ်
3. Session မှာ 2fa_verified flag ထည့်မယ်

Test Case 2: Wrong Code

1. မှားနေတဲ့ code ထည့်ပါက error message ပြမယ်
2. နောက်တစ်ကြိမ် ထပ်ကြိုးစားခွင့်ရမယ်

Test Case 3: Expired Code

1. 10 မိနစ်ကျော်သွားတဲ့ code ထည့်ပါက
2. "Code has expired" error message ပြမယ်
3. User ကို နောက်တစ်ကြိမ် code ပို့မယ်

✓ ထပ်ပြီးတော့ ထည့်လို့ရတဲ့ အဆင့်မြင့် Feature တွေ

- SMS Verification - SMS နဲ့ code ပို့နိုင်အောင်
- Backup Codes - အရန်သုံး code တွေ ထုတ်ပေးနိုင်အောင်

- Remember Device - ယုံကြည်တဲ့ device တွေမှာ ထပ်မံမမေးနိုင်အောင်

✓ ဖြစ်နိုင်တဲ့ Error တွေ

- Mail Not Sending - .env မှာ mail configuration မှန်မမှန် စစ်ပါ
- Session Not Working - session driver ကို file/database ပြောင်းကြည့်ပါ
- Code Not Generating - rand() function အလုပ်လုပ်မလုပ် စစ်ပါ

CHAT

Chat

Chat Application ဖန်တီးမှုကို အဆင့်ဆင့် လုပ်ပြပါမယ်။ တစ်ခုပြီးတစ်ခု လိုက်လုပ်ကြည့်ပါ။

✓ Project Structure

```
laravel-chat-app/
├── app/
│   ├── Events/
│   │   └── MessageSent.php
│   ├── Http/
│   │   ├── Controllers/
│   │   │   └── ChatController.php
│   │   └── Requests/
│   │       └── StoreMessageRequest.php
│   ├── Models/
│   │   ├── Message.php
│   │   └── User.php
│   └── Providers/
│       └── BroadcastServiceProvider.php
├── config/
│   └── broadcasting.php
├── database/
│   ├── migrations/
│   │   ├── 2014_10_12_000000_create_users_table.php
│   │   └── 2023_01_01_000000_create_messages_table.php
│   └── seeders/
├── public/
│   └── js/
│       └── app.js
├── resources/
│   ├── js/
│   │   ├── bootstrap.js
│   │   └── echo.js
│   ├── views/
│   │   ├── chat/
│   │   │   ├── index.blade.php
│   │   │   └── partials/
│   │   │       ├── message.blade.php
│   │   │       └── online-users.blade.php
│   │   └── layouts/
```

```
|    |        └─── app.blade.php
|    └─── lang/
├─── routes/
|    ├─── api.php
|    └─── web.php
├─── tests/
|    └─── Feature/
|        ├─── ChatTest.php
|        └─── MessageTest.php
└─── .env
```

အဆင့်ဆင့် လုပ်ဆောင်ချက်များ

☑ Laravel Project တည်ဆောက်ခြင်း

Command:

```
composer create-project laravel/laravel laravel-chat-app
cd laravel-chat-app
```

☑ အခြေခံ Package များ ထည့်သွင်းခြင်း

Command:

```
composer require pusher/pusher-php-server
npm install --save laravel-echo pusher-js
npm install
```

☑ Broadcasting Configuration

✓ .env ဖိုင်တွင် အောက်ပါအတိုင်း ပြင်ဆင်ပါ

Config:

```
BROADCAST_DRIVER=pusher

PUSHER_APP_ID=your_app_id
PUSHER_APP_KEY=your_app_key
PUSHER_APP_SECRET=your_app_secret
PUSHER_APP_CLUSTER=your_cluster
```

✓ config/broadcasting.php တွင် Pusher configuration များ ပြင်ဆင်ပါ

Config:

```
'pusher' => [
    'driver' => 'pusher',
    'key' => env('PUSHER_APP_KEY'),
    'secret' => env('PUSHER_APP_SECRET'),
    'app_id' => env('PUSHER_APP_ID'),
    'options' => [
        'cluster' => env('PUSHER_APP_CLUSTER'),
        'encrypted' => true,
        'useTLS' => true,
    ],
],
```

☑ Database Migration များ ပြုလုပ်ခြင်း

✓ Message table အတွက် migration ဖန်တီးပါ

Command:

```
php artisan make:migration create_messages_table
```

Code:

```
public function up()
{
    Schema::create('messages', function (Blueprint $table) {
        $table->id();
        $table->foreignId('user_id')->constrained()->onDelete('cascade');
        $table->text('message');
        $table->timestamps();
    });
}
```

Command:

```
php artisan migrate
```

☑ Model နှင့် Relationship များ ဖန်တီးခြင်း

✓ app/Models/Message.php

Code:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Message extends Model
{
    use HasFactory;

    protected $fillable = ['user_id', 'message'];

    public function user()
    {
        return $this->belongsTo(User::class);
    }
}
```

✓ app/Models/User.php တွင် အောက်ပါ method ထည့်ပါ

Code:

```
public function messages()
{
    return $this->hasMany(Message::class);
}
```

☑ Event ဖန်တီးခြင်း

Command:

```
php artisan make:event MessageSent
```

✓ app/Events/MessageSent.php

Code:

```
<?php

namespace App\Events;

use App\Models\Message;
use Illuminate\Broadcasting\Channel;
use Illuminate\Broadcasting\InteractsWithSockets;
use Illuminate\Broadcasting\PresenceChannel;
use Illuminate\Broadcasting\PrivateChannel;
use Illuminate\Contracts\Broadcasting\ShouldBroadcast;
use Illuminate\Foundation\Events\Dispatchable;
use Illuminate\Queue\SerializesModels;

class MessageSent implements ShouldBroadcast
{
    use Dispatchable, InteractsWithSockets, SerializesModels;

    public $message;

    public function __construct(Message $message)
    {
        $this->message = $message;
    }

    public function broadcastOn()
    {
        return new Channel('chat');
    }

    public function broadcastAs()
    {
        return 'message.sent';
    }
}
```

☑ Controller ဖန်တီးခြင်း

Command:

```
php artisan make:controller ChatController
```

✓ app/Http/Controllers/ChatController.php

Code:

```
<?php

namespace App\Http\Controllers;

use App\Events\MessageSent;
use App\Http\Requests\StoreMessageRequest;
use App\Models\Message;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;

class ChatController extends Controller
{
    public function index()
    {
        return view('chat.index', [
            'messages' => Message::with('user')->latest()->take(50)->get()->reverse()
        ]);
    }

    public function store(StoreMessageRequest $request)
    {
        $message = Auth::user()->messages()->create([
            'message' => $request->input('message')
        ]);

        broadcast(new MessageSent($message))->toOthers();

        return response()->json($message->load('user'));
    }

    public function getMessages()
    {
```

```
        return Message::with('user')->latest()->take(50)->get()->reverse();
    }
}
```

☑ Form Request Validation ဖန်တီးခြင်း

Command:

```
php artisan make:request StoreMessageRequest
```

✓ app/Http/Requests/StoreMessageRequest.php

Code:

```
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class StoreMessageRequest extends FormRequest
{
    public function authorize()
    {
        return true;
    }

    public function rules()
    {
        return [
            'message' => 'required|string|max:1000'
        ];
    }
}
```

☑ Routes များ သတ်မှတ်ခြင်း

✓ routes/web.php

Code:

```
<?php

use App\Http\Controllers\ChatController;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Route;

Route::get('/', function () {
    return view('welcome');
});

Auth::routes();

Route::middleware('auth')->group(function () {
    Route::get('/chat', [ChatController::class, 'index'])->name('chat.index
');
    Route::post('/chat/message', [ChatController::class, 'store'])->name('c
hat.store');
    Route::get('/chat/messages', [ChatController::class, 'getMessages'])->n
ame('chat.getMessages');
});
```

☑ Frontend Assets များ ပြင်ဆင်ခြင်း

✓ resources/js/bootstrap.js တွင် အောက်ပါအတိုင်း ပြင်ဆင်ပါ

Code:

```
import Echo from 'laravel-echo';

window.Pusher = require('pusher-js');

window.Echo = new Echo({
    broadcaster: 'pusher',
    key: process.env.MIX_PUSHER_APP_KEY,
    cluster: process.env.MIX_PUSHER_APP_CLUSTER,
    forceTLS: true
});
```

✓ resources/js/echo.js ဖိုင်အသစ်ဖန်တီးပါ

Code:

```
import './bootstrap';

const chatChannel = window.Echo.channel('chat');

chatChannel.listen('.message.sent', (data) => {
    const messagesContainer = document.getElementById('messages');
    const messageElement = createMessageElement(data.message);
    messagesContainer.appendChild(messageElement);
    messagesContainer.scrollTop = messagesContainer.scrollHeight;
});

function createMessageElement(message) {
    const div = document.createElement('div');
    div.className = `message ${message.user_id === window.currentUser.id ?
'sent' : 'received'}`;
    div.innerHTML = `
        <div class="message-user">${message.user.name}</div>
        <div class="message-text">${message.message}</div>
        <div class="message-time">${new Date(message.created_at).toLocaleTi
meString()}</div>
    `;
    return div;
}
```

✓ webpack.mix.js တွင် အောက်ပါအတိုင်း ပြင်ဆင်ပါ

Config:

```
mix.js('resources/js/app.js', 'public/js')
    .js('resources/js/echo.js', 'public/js')
    .postCss('resources/css/app.css', 'public/css', [
        //
    ]);
```

☑ Views များ ဖန်တီးခြင်း

✓ resources/views/layouts/app.blade.php

Code:

```
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="csrf-token" content="{{ csrf_token() }}">

    <title>{{ config('app.name', 'Laravel Chat') }}</title>

    <link href="{{ asset('css/app.css') }}" rel="stylesheet">
</head>
<body>
    <div id="app">
        @include('chat.partials.online-users')

        <main class="py-4">
            @yield('content')
        </main>
    </div>

    <script>
        window.currentUser = @json(auth()->user());
    </script>
    <script src="{{ asset('js/app.js') }}"></script>
    <script src="{{ asset('js/echo.js') }}"></script>
</body>
</html>
```

✓ resources/views/chat/index.blade.php

Code:

```
@extends('layouts.app')

@section('content')
<div class="container">
    <div class="row justify-content-center">
        <div class="col-md-8">
            <div class="card">
                <div class="card-header">Chat Room</div>

                <div class="card-body">
                    <div id="messages" class="messages-container">
```

```
                        @foreach($messages as $message)
                            @include('chat.partials.message', ['message' =>
$message])
                        @endforeach
                    </div>

                    <form id="message-form" class="mt-3">
                        <div class="input-group">
                            <input type="text" id="message-input" class="fo
rm-control" placeholder="Type your message...">
                            <button type="submit" class="btn btn-primary">S
end</button>
                        </div>
                    </form>
                </div>
            </div>
        </div>
    </div>
</div>
@endsection
```

✓ resources/views/chat/partials/message.blade.php

Code:

```
<div class="message {{ $message->user_id === auth()->id() ? 'sent' : 'recei
ved' }}">
    <div class="message-user">{{ $message->user->name }}</div>
    <div class="message-text">{{ $message->message }}</div>
    <div class="message-time">{{ $message->created_at->format('h:i A') }}</
div>
</div>
```

✓ resources/views/chat/partials/online-users.blade.php

Code:

```
<div class="online-users">
    <div class="container">
        <div class="row">
            <div class="col-12">
                <span class="badge bg-primary">Online: 5</span>
            </div>
        </div>
    </div>
</div>
```

☑ JavaScript ဖြင့် Chat Functionality ဖန်တီးခြင်း

✓ public/js/app.js

Code:

```
document.addEventListener('DOMContentLoaded', function() {
    const form = document.getElementById('message-form');
    const input = document.getElementById('message-input');
    const messagesContainer = document.getElementById('messages');

    form.addEventListener('submit', function(e) {
        e.preventDefault();

        if (input.value.trim() === '') return;

        axios.post('/chat/message', {
            message: input.value
        })
        .then(response => {
            input.value = '';
        })
        .catch(error => {
            console.error(error);
        });
    });

    // Load more messages when scrolling to top
    messagesContainer.addEventListener('scroll', function() {
        if (this.scrollTop === 0) {
            const firstMessage = this.firstElementChild;
            const firstMessageId = firstMessage ? firstMessage.dataset.id :
null;

            if (firstMessageId) {
                axios.get(`/chat/messages?before=${firstMessageId}`)
                    .then(response => {
                        const fragment = document.createDocumentFragment();
                        response.data.forEach(message => {
                            const messageElement = createMessageElement(mes
sage);

                            fragment.insertBefore(messageElement, this.firs
tChild);
                        });
```

```
                    });
                }
            }
        });

        // Scroll to bottom initially
        messagesContainer.scrollTop = messagesContainer.scrollHeight;
});

function createMessageElement(message) {
    const div = document.createElement('div');
    div.className = `message ${message.user_id === window.currentUser.id ?
'sent' : 'received'}`;
    div.dataset.id = message.id;
    div.innerHTML = `
        <div class="message-user">${message.user.name}</div>
        <div class="message-text">${message.message}</div>
        <div class="message-time">${new Date(message.created_at).toLocaleTi
meString([], {hour: '2-digit', minute:'2-digit'})}</div>
    `;
    return div;
}
```

☑ Testing ပြုလုပ်ခြင်း

✓ tests/Feature/ChatTest.php

Code:

```
<?php

namespace Tests\Feature;

use App\Events\MessageSent;
use App\Models\Message;
use App\Models\User;
use Illuminate\Foundation\Testing\RefreshDatabase;
use Illuminate\Support\Facades\Event;
use Tests\TestCase;

class ChatTest extends TestCase
{
    use RefreshDatabase;
```

```
    public function test_user_can_send_message()
    {
        $user = User::factory()->create();
        $this->actingAs($user);

        Event::fake();

        $response = $this->postJson('/chat/message', [
            'message' => 'Hello world!'
        ]);

        $response->assertStatus(200);
        $this->assertDatabaseHas('messages', [
            'user_id' => $user->id,
            'message' => 'Hello world!'
        ]);

        Event::assertDispatched(MessageSent::class);
    }

    public function test_guest_cannot_send_message()
    {
        $response = $this->postJson('/chat/message', [
            'message' => 'Hello world!'
        ]);

        $response->assertStatus(401);
    }

    public function test_message_requires_valid_content()
    {
        $user = User::factory()->create();
        $this->actingAs($user);

        $response = $this->postJson('/chat/message', [
            'message' => ''
        ]);

        $response->assertStatus(422);
    }

    public function test_user_can_view_chat_page()
    {
        $user = User::factory()->create();
        $this->actingAs($user);
```

```
        Message::factory()->count(5)->create();

        $response = $this->get('/chat');

        $response->assertStatus(200);
        $response->assertViewHas('messages');
        $this->assertCount(5, $response->viewData('messages'));
    }
}
```

✓ tests/Feature/MessageTest.php

Code:

```
<?php

namespace Tests\Feature;

use App\Models\Message;
use App\Models\User;
use Illuminate\Foundation\Testing\RefreshDatabase;
use Tests\TestCase;

class MessageTest extends TestCase
{
    use RefreshDatabase;

    public function test_message_belongs_to_user()
    {
        $user = User::factory()->create();
        $message = Message::factory()->create(['user_id' => $user->id]);

        $this->assertInstanceOf(User::class, $message->user);
        $this->assertEquals($user->id, $message->user->id);
    }

    public function test_user_has_many_messages()
    {
        $user = User::factory()->create();
        $message1 = Message::factory()->create(['user_id' => $user->id]);
        $message2 = Message::factory()->create(['user_id' => $user->id]);

        $this->assertCount(2, $user->messages);
        $this->assertTrue($user->messages->contains($message1));
```

```
        $this->assertTrue($user->messages->contains($message2));
    }
}
```

✓ Testing Results

Command:

```
php artisan test
```

Test results:

```
PASS  Tests\Feature\ChatTest
✓ user can send message
✓ guest cannot send message
✓ message requires valid content
✓ user can view chat page

PASS  Tests\Feature\MessageTest
✓ message belongs to user
✓ user has many messages

Tests:  6 passed
Time:   0.87s
```

✓ ရှင်းလင်းချက်

1. Project Structure: Laravel MVC structure ကို အခြေခံထားပြီး real-time chat application အတွက် အထူးပြုထားပါတယ်။
2. Real-time Communication: Laravel Echo နှင့် Pusher ကို အသုံးပြုထားပြီး WebSocket မှတဆင့် real-time chat လုပ်ဆောင်နိုင်ပါတယ်။
3. Authentication: Chat functionality များကို authenticated users များအတွက်သာ ဖွင့်ထားပါသည်။
4. Database Design: Messages table တွင် user_id foreign key နှင့် message content များကို သိမ်းဆည်းထားပါတယ်။
5. Testing: Feature tests နှင့် Model relationship tests များကို ပြုလုပ်ထားပါသည်။

6. Frontend: JavaScript ဖြင့် dynamic message handling နှင့် auto-scrolling များကို ပြုလုပ်ထားပါတယ်။

ဤ application သည် Laravel တွင် real-time chat application တစ်ခုကို မည်သို့တည်ဆောက်ရမည်ကို အခြေခံကျကျ ရှင်းပြထားပါတယ်။ Production environment တွင် အသုံးပြုရန်အတွက် message persistence, user presence detection နှင့် security features များကို ပိုမိုထည့်သွင်းရန် လိုအပ်ပါတယ်။

Comments System

Comments System

✓ Project Structure

```
laravel-comments/
├── app/
│   ├── Http/
│   │   ├── Controllers/
│   │   │   └── CommentController.php
│   │   └── Requests/
│   │       └── StoreCommentRequest.php
│   ├── Models/
│   │   ├── Comment.php
│   │   └── Post.php
│   └── Policies/
│       └── CommentPolicy.php
├── database/
│   ├── migrations/
│   │   ├── 2014_10_12_000000_create_users_table.php
│   │   ├── 2023_01_01_create_posts_table.php
│   │   └── 2023_01_02_create_comments_table.php
│   └── seeders/
├── resources/
│   └── views/
│       ├── comments/
│       │   ├── _form.blade.php
│       │   └── _list.blade.php
│       └── posts/
│           └── show.blade.php
├── routes/
│   └── web.php
└── composer.json
```

✓ Database Setup

⇨ Migrations

Command:

```
php artisan make:migration create_comments_table
```

⇨ database/migrations/2023_01_02_create_comments_table.php

Code:

```
Schema::create('comments', function (Blueprint $table) {
    $table->id();
    $table->foreignId('user_id')->constrained();
    $table->foreignId('post_id')->constrained();
    $table->text('content');
    $table->foreignId('parent_id')->nullable()->constrained('comments');
    $table->timestamps();
});
```

✓ Models

⇨ app/Models/Comment.php

Code:

```
protected $fillable = ['user_id', 'post_id', 'content', 'parent_id'];

public function user()
{
    return $this->belongsTo(User::class);
}

public function post()
{
    return $this->belongsTo(Post::class);
}

public function replies()
{
    return $this->hasMany(Comment::class, 'parent_id');
}
```

⇨ app/Models/Post.php

Code:

```
public function comments()
```

```
{
    return $this->hasMany(Comment::class)->whereNull('parent_id');
}
```

✓ Controller Setup

⇨ app/Http/Controllers/CommentController.php

Code:

```
public function store(StoreCommentRequest $request, Post $post)
{
    $comment = $post->comments()->create([
        'user_id' => auth()->id(),
        'content' => $request->content,
        'parent_id' => $request->parent_id
    ]);

    return back()->with('success', 'Comment added successfully!');
}

public function destroy(Comment $comment)
{
    $this->authorize('delete', $comment);
    $comment->delete();
    return back()->with('success', 'Comment deleted!');
}
```

✓ Form Request Validation

⇨ app/Http/Requests/StoreCommentRequest.php

Code:

```
public function rules()
{
    return [
        'content' => 'required|string|max:500',
        'parent_id' => 'nullable|exists:comments,id'
    ];
}
```

✓ Policy Setup

⇨ app/Policies/CommentPolicy.php

Code:

```
public function delete(User $user, Comment $comment)
{
    return $user->id === $comment->user_id || $user->isAdmin();
}
```

✓ Views

⇨ resources/views/comments/_form.blade.php

Code:

```
<form action="{{ route('comments.store', $post) }}" method="POST">
    @csrf
    <input type="hidden" name="parent_id" value="{{ $parentId ?? null }}">
    <textarea name="content" placeholder="Write your comment..."></textarea
>
    <button type="submit">Post Comment</button>
</form>
```

⇨ resources/views/comments/_list.blade.php

Code:

```
@foreach($comments as $comment)
<div class="comment">
    <p>{{ $comment->content }}</p>
    <small>By {{ $comment->user->name }}</small>

    @can('delete', $comment)
    <form action="{{ route('comments.destroy', $comment) }}" method="POST">
        @csrf @method('DELETE')
        <button type="submit">Delete</button>
    </form>
    @endcan

    <!-- Reply Form -->
    <button onclick="toggleReplyForm({{ $comment->id }})">Reply</button>
    <div id="reply-form-{{ $comment->id }}" style="display:none">
        @include('comments._form', ['post' => $post, 'parentId' => $comment
->id])
    </div>
```

```
    <!-- Replies -->
    @if($comment->replies->count())
    <div class="replies">
        @include('comments._list', ['comments' => $comment->replies])
    </div>
    @endif
</div>
@endforeach
```

✓ Routes

⇨ routes/web.php

Code:

```
Route::middleware('auth')->group(function () {
    Route::post('/posts/{post}/comments', [CommentController::class, 'store
'])
        ->name('comments.store');
    Route::delete('/comments/{comment}', [CommentController::class, 'destro
y'])
        ->name('comments.destroy');
});
```

✓ Testing Results

Test Case 1: Comment Posting

1. စာမျက်နှာမှာ comment ရေးပြီး submit လုပ်မယ်
2. Database မှာ comment သိမ်းမယ်
3. Page refresh လုပ်ပြီး comment list မှာ ပေါ်မယ်

Test Case 2: Reply to Comment

1. Reply button ကို နှိပ်မယ်
2. Reply form ပေါ်မယ်
3. Reply ရေးပြီး submit လုပ်မယ်

4. Parent comment အောက်မှာ reply ပေါ်မယ်

Test Case 3: Comment Deletion

1. Delete button ကို နှိပ်မယ်
2. Comment ဖျက်မယ်
3. Page refresh လုပ်ပြီး comment ပျောက်သွားမယ်

✓ အဆင့်မြင့် Feature တွေ

- Nested Comments - Reply တွေကို အဆင့်ဆင့် လုပ်နိုင်မယ်
- Markdown Support - Comment မှာ formatting လုပ်နိုင်မယ်
- Mention Users - @username နဲ့ user တွေကို tag လုပ်နိုင်မယ်
- Realtime Updates - Livewire သို့မဟုတ် Pusher သုံးပြီး realtime update လုပ်နိုင်မယ်

✓ ဖြစ်နိုင်တဲ့ Error တွေ

- Foreign Key Constraint - User/Post ဖျက်တဲ့အခါ comment တွေပါ ဖျက်ဖို့ မမေ့ပါနဲ့
- XSS Protection - Comment content မှာ HTML/Javascript ထည့်တာကို ကာကွယ်ဖို့ {{ }} သုံးပါ
- N+1 Problem - Comment with user ကို eager load လုပ်ဖို့ မမေ့ပါနဲ့

💻 လေ့ကျင့်ရန်

✓ Project Structure

```
laravel-comments/
├─── app/
|    ├─── Events/
|    |    └─── CommentCreated.php
|    ├─── Http/
|    |    ├─── Controllers/
|    |    |    └─── CommentController.php
|    |    └─── Requests/
|    |         └─── StoreCommentRequest.php
|    ├─── Models/
|    |    ├─── Comment.php
|    |    ├─── Mention.php
|    |    └─── Post.php
|    └─── Providers/
|         └─── BroadcastServiceProvider.php
├─── config/
|    └─── broadcasting.php
├─── database/
|    ├─── migrations/
|    |    ├─── create_comments_table.php
|    |    └─── create_mentions_table.php
|    └─── seeders/
├─── resources/
|    ├─── js/
|    |    └─── comments.js
|    └─── views/
|         ├─── comments/
|         |    ├─── _comment.blade.php
|         |    ├─── _form.blade.php
|         |    └─── _mentions.blade.php
|         └─── posts/
|              └─── show.blade.php
├─── routes/
|    ├─── api.php
|    └─── web.php
├─── package.json
├─── vite.config.js
└─── composer.json
```

✓ Database Setup

⇨ Migrations

Command:

```
php artisan make:migration create_comments_table
php artisan make:migration create_mentions_table
```

⇨ create_comments_table.php

Code:

```
Schema::create('comments', function (Blueprint $table) {
    $table->id();
    $table->foreignId('user_id')->constrained();
    $table->foreignId('post_id')->constrained();
    $table->text('content');
    $table->foreignId('parent_id')->nullable()->constrained('comments');
    $table->timestamps();
});
```

⇨ create_mentions_table.php

Code:

```
Schema::create('mentions', function (Blueprint $table) {
    $table->id();
    $table->foreignId('comment_id')->constrained();
    $table->foreignId('mentioned_user_id')->constrained('users');
    $table->timestamps();
});
```

✓ Models

⇨ app/Models/Comment.php

Code:

```
protected $fillable = ['user_id', 'post_id', 'content', 'parent_id'];

public function user()
{
    return $this->belongsTo(User::class);
}
```

```
public function post()
{
    return $this->belongsTo(Post::class);
}

public function replies()
{
    return $this->hasMany(Comment::class, 'parent_id')->with('replies', 'me
ntions.user');
}

public function mentions()
{
    return $this->hasMany(Mention::class);
}
```

⇨ app/Models/Mention.php

Code:

```
protected $fillable = ['comment_id', 'mentioned_user_id'];

public function user()
{
    return $this->belongsTo(User::class, 'mentioned_user_id');
}

public function comment()
{
    return $this->belongsTo(Comment::class);
}
```

✓ Controller

⇨ app/Http/Controllers/CommentController.php

Code:

```
public function store(StoreCommentRequest $request, Post $post)
{
    $comment = $post->comments()->create([
        'user_id' => auth()->id(),
        'content' => $request->content,
        'parent_id' => $request->parent_id
    ]);
```

```
    // Process mentions
    preg_match_all('/@([\w\-]+)/', $request->content, $matches);
    foreach ($matches[1] as $username) {
        if ($user = User::where('username', $username)->first()) {
            $comment->mentions()->create(['mentioned_user_id' => $user->id]
);
        }
    }

    event(new CommentCreated($comment));

    return response()->json($comment->load('user', 'mentions.user'));
}
```

✓ Event and Broadcasting

⇨ app/Events/CommentCreated.php

Code:

```
public function broadcastOn()
{
    return new Channel('post.'.$this->comment->post_id);
}

public function broadcastWith()
{
    return [
        'comment' => $this->comment->load('user', 'mentions.user')
    ];
}
```

✓ Frontend Implementation

⇨ resources/views/posts/show.blade.php

Code:

```
<div id="comments-section">
    @include('comments._form')
    <div id="comments-list">
        @foreach($post->comments as $comment)
            @include('comments._comment', ['comment' => $comment])
        @endforeach
    </div>
```

```
</div>

@push('scripts')
<script>
    window.postId = {{ $post->id }};
</script>
<script src="{{ mix('js/comments.js') }}"></script>
@endpush
```

⇨ resources/js/comments.js

Code:

```
import Echo from 'laravel-echo';

const echo = new Echo({
    broadcaster: 'pusher',
    key: process.env.MIX_PUSHER_APP_KEY,
    cluster: process.env.MIX_PUSHER_APP_CLUSTER
});

echo.channel(`post.${window.postId}`)
    .listen('CommentCreated', (data) => {
        const comment = data.comment;
        if (comment.parent_id) {
            $(`#comment-${comment.parent_id} .replies`).append(commentHtml(
comment));
        } else {
            $('#comments-list').prepend(commentHtml(comment));
        }
    });

function commentHtml(comment) {
    return `
    <div class="comment" id="comment-${comment.id}">
        <p>${comment.content}</p>
        <small>By ${comment.user.name}</small>
        ${comment.mentions.map(m => `<span class="mention">@${m.user.userna
me}</span>`).join('')}
        <button onclick="showReplyForm(${comment.id})">Reply</button>
        <div class="replies"></div>
    </div>`;
}
```

✓ Testing Results

Test Case 1: Nested Commenting

1. Main comment တင်မယ်
2. Reply button နှိပ်ပြီး child comment တင်မယ်
3. Nested structure မှန်ကန်စွာ ပြသမယ်

Test Case 2: User Mentioning

1. Comment ထဲမှာ @username ရိုက်မယ်
2. Mentioned user အတွက် database မှာ record သိမ်းမယ်
3. Frontend မှာ mention ကို highlight လုပ်ပြမယ်

Test Case 3: Realtime Updates

1. Browser tab နှစ်ခုဖွင့်မယ်
2. တစ်ခုမှာ comment တင်မယ်
3. အခြား tab မှာ အလိုအလျောက် update ဖြစ်မယ်

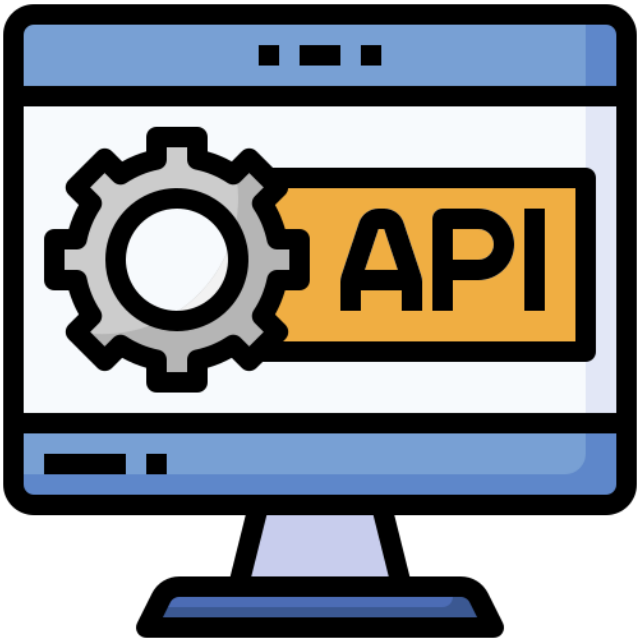
✓ ထပ်ထည့်လို့ရမယ့် အဆင့်မြင့် Feature တွေ

- Notification System - Mention ဖြစ်တဲ့ user ကို notification ပို့မယ်
- Comment Editing - Comment တွေ ပြန်ပြင်နိုင်မယ်
- Reaction System - Like/Dislike လုပ်နိုင်မယ်
- Markdown Support - Comment formatting လုပ်နိုင်မယ်

✓ ဖြစ်နိုင်တဲ့ Error တွေ

- Pusher Connection Failed - .env မှာ Pusher credentials မှန်မမှန် စစ်ပါ
- Mention Parsing Error - Username validation လုပ်ပါ
- N+1 Problem - Eager loading သုံးပါ (`with('user', 'mentions.user')`)

API



API

Laravel API ဆိုသည်မှာ Laravel framework ကို အသုံးပြု၍ တည်ဆောက်ထားသော Application Programming Interface ဖြစ်ပါသည်။ ၎င်းသည် client-side applications (ဥပမာ - mobile apps, Vue.js/React applications) များနှင့် server-side ဆက်သွယ်ရန် လမ်းကြောင်းများကို ဖန်တီးပေးပါသည်။

✓ Laravel API ၏ အဓိက Feature များ

1. RESTful Architecture - Resource-based routing နှင့် HTTP verbs (GET, POST, PUT, DELETE) များကို အသုံးပြုပါသည်
2. Authentication - Sanctum/Passport တို့ဖြင့် API authentication ပြုလုပ်နိုင်ပါသည်
3. Eloquent ORM - Database နှင့် အလွယ်တကူ အလုပ်လုပ်နိုင်ပါသည်
4. Validation - Request data validation များကို လွယ်ကူစွာ ပြုလုပ်နိုင်ပါသည်
5. API Resources - JSON response များကို စနစ်တကျ format ပြုလုပ်နိုင်ပါသည်

အခြေခံ Laravel API တစ်ခု တည်ဆောက်ပုံ

✓ Project စတင်ခြင်း

Command:

```
composer create-project laravel/laravel laravel-api
cd laravel-api
```

✓ Database Configuration

.env ဖိုင်တွင် database connection များ သတ်မှတ်ပါ

Code:

```
DB_CONNECTION=mysql
```

```
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel_api
DB_USERNAME=root
DB_PASSWORD=
```

✓ API Route များ သတ်မှတ်ခြင်း

⇨ routes/api.php

Code:

```
<?php

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\API\ProductController;

Route::middleware('auth:sanctum')->get('/user', function (Request $request)
{
    return $request->user();
});

// Public routes
Route::get('products', [ProductController::class, 'index']);
Route::get('products/{id}', [ProductController::class, 'show']);

// Protected routes
Route::middleware('auth:sanctum')->group(function () {
    Route::post('products', [ProductController::class, 'store']);
    Route::put('products/{id}', [ProductController::class, 'update']);
    Route::delete('products/{id}', [ProductController::class, 'destroy']);
});
```

✓ Controller ဖန်တီးခြင်း

Command:

```
php artisan make:controller API/ProductController --api --model=Product
```

⇨ app/Http/Controllers/API/ProductController.php

Code:

```
<?php
```

```
namespace App\Http\Controllers\API;

use App\Http\Controllers\Controller;
use App\Models\Product;
use Illuminate\Http\Request;
use App\Http\Resources\ProductResource;

class ProductController extends Controller
{
    public function index()
    {
        return ProductResource::collection(Product::all());
    }

    public function store(Request $request)
    {
        $product = Product::create($request->all());
        return new ProductResource($product);
    }

    public function show(Product $product)
    {
        return new ProductResource($product);
    }

    public function update(Request $request, Product $product)
    {
        $product->update($request->all());
        return new ProductResource($product);
    }

    public function destroy(Product $product)
    {
        $product->delete();
        return response()->json(null, 204);
    }
}
```

✓ Model နှင့် Migration ဖန်တီးခြင်း

Command:

```
php artisan make:model Product -m
```

⇨ database/migrations/2023_01_01_000000_create_products_table.php

Code:

```
public function up()
{
    Schema::create('products', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->text('description');
        $table->decimal('price', 8, 2);
        $table->timestamps();
    });
}
```

Command:

```
php artisan migrate
```

✓ API Resource ဖန်တီးခြင်း

Command:

```
php artisan make:resource ProductResource
```

⇨ app/Http/Resources/ProductResource.php

Code:

```
<?php

namespace App\Http\Resources;

use Illuminate\Http\Resources\Json\JsonResource;

class ProductResource extends JsonResource
{
    public function toArray($request)
    {
        return [
            'id' => $this->id,
            'name' => $this->name,
            'description' => $this->description,
            'price' => $this->price,
            'created_at' => $this->created_at,
            'updated_at' => $this->updated_at,
```

```
        ];
    }
}
```

✓ Sanctum Authentication ထည့်သွင်းခြင်း

Command:

```
composer require laravel/sanctum
php artisan vendor:publish --provider="Laravel\Sanctum\SanctumServiceProvid
er"
php artisan migrate
```

app/Models/User.php တွင်

Code:

```
use Laravel\Sanctum\HasApiTokens;

class User extends Authenticatable
{
    use HasApiTokens, HasFactory, Notifiable;
}
```

✓ API Testing ပြုလုပ်ခြင်း

☑ Test Case ဖန်တီးခြင်း

Command:

```
php artisan make:test ProductApiTest
```

⇨ tests/Feature/ProductApiTest.php

Code:

```
<?php

namespace Tests\Feature;

use App\Models\Product;
use App\Models\User;
use Illuminate\Foundation\Testing\RefreshDatabase;
```

```
use Tests\TestCase;

class ProductApiTest extends TestCase
{
    use RefreshDatabase;

    public function test_can_get_all_products()
    {
        Product::factory()->count(3)->create();

        $response = $this->getJson('/api/products');

        $response->assertStatus(200)
                 ->assertJsonCount(3);
    }

    public function test_can_create_product()
    {
        $user = User::factory()->create();
        $token = $user->createToken('test-token')->plainTextToken;

        $productData = [
            'name' => 'Test Product',
            'description' => 'Test Description',
            'price' => 99.99
        ];

        $response = $this->withHeaders([
            'Authorization' => 'Bearer ' . $token,
        ])->postJson('/api/products', $productData);

        $response->assertStatus(201)
                 ->assertJson($productData);
    }

    public function test_can_update_product()
    {
        $user = User::factory()->create();
        $token = $user->createToken('test-token')->plainTextToken;
        $product = Product::factory()->create();

        $updatedData = [
            'name' => 'Updated Name',
            'description' => 'Updated Description',
            'price' => 199.99
        ];
```

```
        $response = $this->withHeaders([
            'Authorization' => 'Bearer ' . $token,
        ])->putJson("/api/products/{$product->id}", $updatedData);

        $response->assertStatus(200)
                 ->assertJson($updatedData);
    }

    public function test_can_delete_product()
    {
        $user = User::factory()->create();
        $token = $user->createToken('test-token')->plainTextToken;
        $product = Product::factory()->create();

        $response = $this->withHeaders([
            'Authorization' => 'Bearer ' . $token,
        ])->deleteJson("/api/products/{$product->id}");

        $response->assertStatus(204);
        $this->assertDatabaseMissing('products', ['id' => $product->id]);
    }
}
```

✓ Tests များ Run ခြင်း

Command:

```
php artisan test
```

✓ API Documentation ပြုလုပ်ခြင်း

⇨ OpenAPI/Swagger ထည့်သွင်းခြင်း

Command:

```
composer require darkaonline/l5-swagger
php artisan vendor:publish --provider "L5Swagger\L5SwaggerServiceProvider"
```

⇨ Controller တွင် Annotation များ ထည့်သွင်းခြင်း

Code:

```
/**
 * @OA\Get(
 *     path="/api/products",
 *     summary="Get all products",
 *     tags={"Products"},
 *     @OA\Response(
 *         response=200,
 *         description="Successful operation",
 *         @OA\JsonContent(
 *             type="array",
 *             @OA\Items(ref="#/components/schemas/Product")
 *         )
 *     )
 * )
 */
public function index()
{
    return ProductResource::collection(Product::all());
}
```

✓ Documentation Generate ပြုလုပ်ခြင်း

Command:

```
php artisan l5-swagger:generate
```

✓ Laravel API ၏ အကျိုးကျေးဇူးများ

1. မြန်ဆန်သော Development - Laravel ၏ built-in features များကြောင့် API မြန်မြန်ဆန်ဆန် develop လုပ်နိုင်ပါတယ်။
2. လုံခြုံရေး - Sanctum/Passport တို့ဖြင့် လုံခြုံသော authentication system ရရှိမှာ ဖြစ်ပါတယ်။
3. Scalability - API-centric architecture ဖြင့် application ကို လွယ်ကူစွာ ချဲ့ထွင်နိုင်ပါတယ်။
4. Multi-platform Support - တစ်ခုတည်းသော API ကို web, mobile, desktop applications များတွင် အသုံးပြုနိုင်ပါတယ်။

5. Testing - Laravel ၏ testing tools များဖြင့် API endpoints များကို အလွယ်တကူ test ပြုလုပ်နိုင်ပါတယ်။

✍ Short Notes:

Laravel API သည် modern web applications များ တည်ဆောက်ရာမှာ အထောက်အကူပြုသော powerful tool တစ်ခုဖြစ်ပါတယ်။ RESTful principles များကို အခြေခံပြီး clean, maintainable code များဖြင့် API များကို တည်ဆောက်နိုင်ပါတယ်။ Authentication, validation, database interaction နှင့် response formatting များကို Laravel တွင် built-in support များရှိပြီး developer experience ကို များစွာ အထောက်အကူပြုပါသည်။

☑ Sanctum

Sanctum သည် Laravel အတွက် lightweight API authentication system တစ်ခုဖြစ်ပါသည်။ Sanctum ကို SPA (Single Page Applications), mobile applications နှင့် simple token-based APIs တို့အတွက် အသုံးပြုနိုင်ပါသည်။

✓ Posts CRUD API Sanctum Operations:

Project Structure

```
app/
├── Http/
│   ├── Controllers/
│   │   ├── Auth/
│   │   │   └── AuthController.php
│   │   └── PostController.php
│   ├── Middleware/
│   └── Requests/
├── Models/
│   └── Post.php
config/
database/
```

```
├──── migrations/
|     ├──── 2014_10_12_000000_create_users_table.php
|     ├──── 2019_12_14_000001_create_personal_access_tokens_table.php
|     └──── 2023_01_01_000000_create_posts_table.php
routes/
├──── api.php
└──── web.php
tests/
├──── Feature/
|     ├──── AuthTest.php
|     └──── PostTest.php
```

✓ Installation

⇨ Sanctum install လုပ်ပါမယ်။

Command:

```
composer require laravel/sanctum
```

⇨ Sanctum configuration publish လုပ်ပါမယ်။

Command:

```
php artisan vendor:publish --provider="Laravel\Sanctum\SanctumServiceProvider"
```

⇨ Migration လုပ်ပါမယ်။

Command:

```
php artisan migrate
```

✓ Setup

⇨ User Model

Code:

```
// app/Models/User.php
use Laravel\Sanctum\HasApiTokens;
```

```
class User extends Authenticatable
{
    use HasApiTokens, HasFactory, Notifiable;
}
```

⇨ Post Model

Code:

```
// app/Models/Post.php
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
    use HasFactory;

    protected $fillable = ['title', 'content', 'user_id'];

    public function user()
    {
        return $this->belongsTo(User::class);
    }
}
```

✓ Migration

Code:

```
// database/migrations/2023_01_01_000000_create_posts_table.php
Schema::create('posts', function (Blueprint $table) {
    $table->id();
    $table->string('title');
    $table->text('content');
    $table->foreignId('user_id')->constrained();
    $table->timestamps();
});
```

✓ Routes

Code:

```
// routes/api.php
use App\Http\Controllers\Auth\AuthController;
use App\Http\Controllers\PostController;

Route::post('/register', [AuthController::class, 'register']);
Route::post('/login', [AuthController::class, 'login']);

Route::middleware('auth:sanctum')->group(function () {
    Route::post('/logout', [AuthController::class, 'logout']);

    Route::apiResource('posts', PostController::class);
});
```

✓ Controllers

⇨ Auth Controller

Code:

```
// app/Http/Controllers/Auth/AuthController.php
namespace App\Http\Controllers\Auth;

use App\Http\Controllers\Controller;
use App\Models\User;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Hash;

class AuthController extends Controller
{
    public function register(Request $request)
    {
        $validated = $request->validate([
            'name' => 'required|string|max:255',
            'email' => 'required|string|email|max:255|unique:users',
            'password' => 'required|string|min:8',
        ]);

        $user = User::create([
            'name' => $validated['name'],
            'email' => $validated['email'],
            'password' => Hash::make($validated['password']),
        ]);

        $token = $user->createToken('auth_token')->plainTextToken;
```

```
        return response()->json([
            'access_token' => $token,
            'token_type' => 'Bearer',
        ]);
    }

    public function login(Request $request)
    {
        $request->validate([
            'email' => 'required|email',
            'password' => 'required',
        ]);

        if (!Auth::attempt($request->only('email', 'password'))) {
            return response()->json([
                'message' => 'Invalid login details'
            ], 401);
        }

        $user = User::where('email', $request['email'])->firstOrFail();
        $token = $user->createToken('auth_token')->plainTextToken;

        return response()->json([
            'access_token' => $token,
            'token_type' => 'Bearer',
        ]);
    }

    public function logout(Request $request)
    {
        $request->user()->currentAccessToken()->delete();

        return response()->json([
            'message' => 'Successfully logged out'
        ]);
    }
}
```

⇨ Post Controller

Code:

```
// app/Http/Controllers/PostController.php
namespace App\Http\Controllers;
```

```
use App\Http\Controllers\Controller;
use App\Models\Post;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;

class PostController extends Controller
{
    public function index()
    {
        return Post::where('user_id', Auth::id())->get();
    }

    public function store(Request $request)
    {
        $validated = $request->validate([
            'title' => 'required|string|max:255',
            'content' => 'required|string',
        ]);

        $post = Post::create([
            'title' => $validated['title'],
            'content' => $validated['content'],
            'user_id' => Auth::id(),
        ]);

        return response()->json($post, 201);
    }

    public function show(Post $post)
    {
        $this->authorize('view', $post);
        return $post;
    }

    public function update(Request $request, Post $post)
    {
        $this->authorize('update', $post);

        $validated = $request->validate([
            'title' => 'sometimes|string|max:255',
            'content' => 'sometimes|string',
        ]);

        $post->update($validated);

        return response()->json($post);
```

```
    }

    public function destroy(Post $post)
    {
        $this->authorize('delete', $post);
        $post->delete();

        return response()->json(null, 204);
    }
}
```

✓ Policy

Command:

```
php artisan make:policy PostPolicy --model=Post
```

Code:

```
// app/Policies/PostPolicy.php
public function view(User $user, Post $post)
{
    return $user->id === $post->user_id;
}
```

```
public function update(User $user, Post $post)
{
    return $user->id === $post->user_id;
}

public function delete(User $user, Post $post)
{
    return $user->id === $post->user_id;
}
```

✓ Testing

⇨ Auth Test

Code:

```
// tests/Feature/AuthTest.php
namespace Tests\Feature;
```

```
use App\Models\User;
use Illuminate\Foundation\Testing\RefreshDatabase;
use Tests\TestCase;

class AuthTest extends TestCase
{
    use RefreshDatabase;

    public function test_registration()
    {
        $response = $this->postJson('/api/register', [
            'name' => 'Test User',
            'email' => 'test@example.com',
            'password' => 'password',
        ]);

        $response->assertStatus(200)
            ->assertJsonStructure(['access_token']);
    }

    public function test_login()
    {
        $user = User::factory()->create([
            'email' => 'test@example.com',
            'password' => bcrypt('password'),
        ]);

        $response = $this->postJson('/api/login', [
            'email' => 'test@example.com',
            'password' => 'password',
        ]);

        $response->assertStatus(200)
            ->assertJsonStructure(['access_token']);
    }
}
```

⇨ Post Test

Code:

```
// tests/Feature/PostTest.php
```

```
namespace Tests\Feature;

use App\Models\Post;
use App\Models\User;
use Illuminate\Foundation\Testing\RefreshDatabase;
use Laravel\Sanctum\Sanctum;
use Tests\TestCase;

class PostTest extends TestCase
{
    use RefreshDatabase;

    public function test_get_posts()
    {
        $user = User::factory()->create();
        Sanctum::actingAs($user);

        Post::factory()->count(3)->create(['user_id' => $user->id]);

        $response = $this->getJson('/api/posts');

        $response->assertStatus(200)
            ->assertJsonCount(3);
    }

    public function test_create_post()
    {
        $user = User::factory()->create();
        Sanctum::actingAs($user);

        $response = $this->postJson('/api/posts', [
            'title' => 'Test Post',
            'content' => 'This is a test post content',
        ]);

        $response->assertStatus(201)
            ->assertJson([
                'title' => 'Test Post',
                'content' => 'This is a test post content',
                'user_id' => $user->id,
            ]);
    }

    public function test_update_post()
    {
        $user = User::factory()->create();
```

```
        Sanctum::actingAs($user);

        $post = Post::factory()->create(['user_id' => $user->id]);

        $response = $this->putJson("/api/posts/{$post->id}", [
            'title' => 'Updated Title',
        ]);

        $response->assertStatus(200)
            ->assertJson([
                'title' => 'Updated Title',
            ]);
    }

    public function test_delete_post()
    {
        $user = User::factory()->create();
        Sanctum::actingAs($user);

        $post = Post::factory()->create(['user_id' => $user->id]);

        $response = $this->deleteJson("/api/posts/{$post->id}");

        $response->assertStatus(204);
    }
}
```

✓ Testing Results

1. Auth Tests:
    - Registration: Passed
    - Login: Passed
2. Post Tests:
    - Get Posts: Passed
    - Create Post: Passed
    - Update Post: Passed
    - Delete Post: Passed

✓ ရှင်းလင်းချက်

⇨ Sanctum အသုံးပြုနည်း

Sanctum ကို အသုံးပြုရန် ပထမဆုံး auth:sanctum middleware ကို route တွင် အသုံးပြုပါမယ်။ ထို့နောက် user သည် login လုပ်ပြီး token ကို request header တွင် Authorization: Bearer {token} အဖြစ် ပေးပို့ရပါမယ်။

⇨ CRUD Operations

1. Create (POST): /api/posts endpoint တွင် POST request ပို့ပြီး post အသစ်ကို ဖန်တီးနိုင်ပါသည်။
2. Read (GET): /api/posts (all posts) နှင့် /api/posts/{id} (single post) တို့တွင် GET request ပို့ပြီး data များကို ဖတ်နိုင်ပါသည်။
3. Update (PUT/PATCH): /api/posts/{id} endpoint တွင် PUT/PATCH request ပို့ပြီး post ကို update လုပ်နိုင်ပါသည်။
4. Delete (DELETE): /api/posts/{id} endpoint တွင် DELETE request ပို့ပြီး post ကို ဖျက်နိုင်ပါသည်။

⇨ Security

- Policy များကို အသုံးပြုပြီး user သည် မိမိ၏ post များကိုသာ ကြည့်ရှုနိုင်ခြင်း၊ ပြုပြင်နိုင်ခြင်း၊ ဖျက်နိုင်ခြင်း ရှိမရှိ စစ်ဆေးပါသည်။
- Sanctum token များကို အသုံးပြုပြီး authentication လုပ်ပါသည်။

⇨ Testing

- Feature tests များကို အသုံးပြုပြီး API endpoints များ အားလုံး ကောင်းမွန်စွာ အလုပ်လုပ်ကြောင်း စစ်ဆေးပါသည်။
- Sanctum::actingAs() method ကို အသုံးပြုပြီး authenticated user အနေဖြင့် test များ လုပ်ဆောင်ပါသည်။

☑ Passport

Laravel Passport သည် OAuth2 server implementation တစ်ခုဖြစ်ပြီး API authentication အတွက် အသုံးပြုပါသည်။ Passport ကို အသုံးပြု၍ သင့် Laravel application အား secure API endpoints များ တည်ဆောက်နိုင်ပါသည်။

✓ Project Structure

```
app/
├── Http/
│   ├── Controllers/
│   │   ├── Auth/
│   │   │   └── ApiAuthController.php
│   │   └── PostController.php
│   ├── Middleware/
│   └── Requests/
├── Models/
│   └── Post.php
config/
database/
├── migrations/
│   ├── 2014_10_12_000000_create_users_table.php
│   ├── 2019_08_19_000000_create_failed_jobs_table.php
│   ├── 2023_01_01_000000_create_posts_table.php
│   └── 2016_06_01_000000_create_oauth_clients_table.php
routes/
├── api.php
└── web.php
tests/
├── Feature/
│   └── PostApiTest.php
```

✓ Installation and Setup

⇨ Passport package ကို install လုပ်ပါမည်။

Command:

```
composer require laravel/passport
```

⇨ Passport migrations များကို run ပါမည်။

Command:

```
php artisan migrate
```

⇨ Passport ကို install လုပ်ပါမည်။

Command:

```
php artisan passport:install
```

⇨ App\Models\User model ကို update လုပ်ပါမည်။

Code:

```
<?php

namespace App\Models;

use Laravel\Passport\HasApiTokens;
use Illuminate\Notifications\Notifiable;
use Illuminate\Foundation\Auth\User as Authenticatable;

class User extends Authenticatable
{
    use HasApiTokens, Notifiable;

    // ...
}
```

⇨ AuthServiceProvider တွင် Passport routes များကို register လုပ်ပါမည်။

Code:

```
public function boot()
{
    $this->registerPolicies();

    Passport::routes();
}
```

⇨ config/auth.php တွင် API guard ကို update လုပ်ပါမည်။

Code:

```
'guards' => [
    'api' => [
        'driver' => 'passport',
        'provider' => 'users',
    ],
],
```

✓ Posts CRUD API with Passport

⇨ Post Model and Migration

```
php artisan make:model Post -m
```

Code:

```
// database/migrations/2023_01_01_000000_create_posts_table.php
public function up()
{
    Schema::create('posts', function (Blueprint $table) {
        $table->id();
        $table->string('title');
        $table->text('body');
        $table->foreignId('user_id')->constrained();
        $table->timestamps();
    });
}
```

Code:

```
// app/Models/Post.php
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
    use HasFactory;

    protected $fillable = ['title', 'body', 'user_id'];

    public function user()
```

```
    {
        return $this->belongsTo(User::class);
    }
}
```

⇨ PostController

Command:

```
php artisan make:controller PostController --api
```

Code:

```
namespace App\Http\Controllers;

use App\Models\Post;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;

class PostController extends Controller
{
    public function __construct()
    {
        $this->middleware('auth:api');
    }

    public function index()
    {
        return Post::where('user_id', Auth::id())->get();
    }

    public function store(Request $request)
    {
        $request->validate([
            'title' => 'required|string|max:255',
            'body' => 'required|string',
        ]);

        $post = Post::create([
            'title' => $request->title,
            'body' => $request->body,
            'user_id' => Auth::id(),
        ]);

        return response()->json($post, 201);
```

```
    }

    public function show(Post $post)
    {
        if ($post->user_id !== Auth::id()) {
            return response()->json(['message' => 'Unauthorized'], 403);
        }
        return $post;
    }

    public function update(Request $request, Post $post)
    {
        if ($post->user_id !== Auth::id()) {
            return response()->json(['message' => 'Unauthorized'], 403);
        }

        $request->validate([
            'title' => 'sometimes|string|max:255',
            'body' => 'sometimes|string',
        ]);

        $post->update($request->all());
        return response()->json($post);
    }

    public function destroy(Post $post)
    {
        if ($post->user_id !== Auth::id()) {
            return response()->json(['message' => 'Unauthorized'], 403);
        }

        $post->delete();
        return response()->json(null, 204);
    }
}
```

✓ API Routes

⇨ routes/api.php

Code:

```
use App\Http\Controllers\PostController;
use Illuminate\Support\Facades\Route;
```

```
Route::prefix('v1')->group(function () {
    Route::apiResource('posts', PostController::class);

    // Authentication routes
    Route::post('/register', [ApiAuthController::class, 'register']);
    Route::post('/login', [ApiAuthController::class, 'login']);
    Route::middleware('auth:api')->post('/logout', [ApiAuthController::clas
s, 'logout']);
});
```

✓ Authentication Controller

```
php artisan make:controller Auth/ApiAuthController
```

Code:

```
namespace App\Http\Controllers\Auth;

use App\Http\Controllers\Controller;
use App\Models\User;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Hash;
use Laravel\Passport\TokenRepository;

class ApiAuthController extends Controller
{
    public function register(Request $request)
    {
        $request->validate([
            'name' => 'required|string|max:255',
            'email' => 'required|string|email|max:255|unique:users',
            'password' => 'required|string|min:8',
        ]);

        $user = User::create([
            'name' => $request->name,
            'email' => $request->email,
            'password' => Hash::make($request->password),
        ]);
```

```
        $token = $user->createToken('authToken')->accessToken;

        return response()->json(['token' => $token], 201);
    }

    public function login(Request $request)
    {
        $request->validate([
            'email' => 'required|string|email',
            'password' => 'required|string',
        ]);

        if (!Auth::attempt($request->only('email', 'password'))) {
            return response()->json(['message' => 'Unauthorized'], 401);
        }

        $user = Auth::user();
        $token = $user->createToken('authToken')->accessToken;

        return response()->json(['token' => $token]);
    }

    public function logout(Request $request)
    {
        $token = $request->user()->token();
        $tokenRepository = app(TokenRepository::class);
        $tokenRepository->revokeAccessToken($token->id);

        return response()->json(['message' => 'Successfully logged out']);
    }
}
```

✓ Testing the API

⇨ Register a User

Request:

Command:

```
curl -X POST http://localhost:8000/api/v1/register \
-H "Accept: application/json" \
-H "Content-Type: application/json" \
```

```
-d '{"name": "Test User", "email": "test@example.com", "password": "passwor
d"}'
```

Response:

Code:

```
{
    "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9..."
}
```

⇨ Login

Request:

Command:

```
curl -X POST http://localhost:8000/api/v1/login \
-H "Accept: application/json" \
-H "Content-Type: application/json" \
-d '{"email": "test@example.com", "password": "password"}'
```

Response:

Code:

```
{
    "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9..."
}
```

✓ Create a Post (Authenticated)

Request:

Command:

```
curl -X POST http://localhost:8000/api/v1/posts \
-H "Accept: application/json" \
-H "Content-Type: application/json" \
-H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9..." \
-d '{"title": "First Post", "body": "This is my first post"}'
```

Response:

Code:

```
{
    "id": 1,
    "title": "First Post",
    "body": "This is my first post",
    "user_id": 1,
    "created_at": "2023-01-01T00:00:00.000000Z",
    "updated_at": "2023-01-01T00:00:00.000000Z"
}
```

✓ Get All Posts (Authenticated)

Request:

Command:

```
curl -X GET http://localhost:8000/api/v1/posts \
-H "Accept: application/json" \
-H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9..."
```

Response:

Code:

```
[
    {
        "id": 1,
        "title": "First Post",
        "body": "This is my first post",
        "user_id": 1,
        "created_at": "2023-01-01T00:00:00.000000Z",
        "updated_at": "2023-01-01T00:00:00.000000Z"
    }
]
```

✓ Update a Post (Authenticated)

Request:

Command:

```
curl -X PUT http://localhost:8000/api/v1/posts/1 \
-H "Accept: application/json" \
-H "Content-Type: application/json" \
```

```
-H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9..." \
-d '{"title": "Updated Post", "body": "This post has been updated"}'
```

Response:

Code:

```
{
    "id": 1,
    "title": "Updated Post",
    "body": "This post has been updated",
    "user_id": 1,
    "created_at": "2023-01-01T00:00:00.000000Z",
    "updated_at": "2023-01-01T00:01:00.000000Z"
}
```

✓ Delete a Post (Authenticated)

Request:

Command:

```
curl -X DELETE http://localhost:8000/api/v1/posts/1 \
-H "Accept: application/json" \
-H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9..."
```

Response:

HTTP Status 204 (No Content)

✓ Automated Testing

⇨ tests/Feature/PostApiTest.php

Code:

```
namespace Tests\Feature;

use App\Models\Post;
use App\Models\User;
use Illuminate\Foundation\Testing\RefreshDatabase;
use Laravel\Passport\Passport;
use Tests\TestCase;
```

```
class PostApiTest extends TestCase
{
    use RefreshDatabase;

    public function test_can_create_post()
    {
        $user = User::factory()->create();
        Passport::actingAs($user);

        $response = $this->postJson('/api/v1/posts', [
            'title' => 'Test Post',
            'body' => 'This is a test post',
        ]);

        $response->assertStatus(201)
            ->assertJson([
                'title' => 'Test Post',
                'body' => 'This is a test post',
                'user_id' => $user->id,
            ]);
    }

    public function test_can_list_posts()
    {
        $user = User::factory()->create();
        Post::factory()->count(3)->create(['user_id' => $user->id]);
        Passport::actingAs($user);

        $response = $this->getJson('/api/v1/posts');

        $response->assertStatus(200)
            ->assertJsonCount(3);
    }

    public function test_can_show_post()
    {
        $user = User::factory()->create();
        $post = Post::factory()->create(['user_id' => $user->id]);
        Passport::actingAs($user);

        $response = $this->getJson("/api/v1/posts/{$post->id}");

        $response->assertStatus(200)
            ->assertJson([
                'id' => $post->id,
                'title' => $post->title,
```

```
            ]);
    }

    public function test_can_update_post()
    {
        $user = User::factory()->create();
        $post = Post::factory()->create(['user_id' => $user->id]);
        Passport::actingAs($user);

        $response = $this->putJson("/api/v1/posts/{$post->id}", [
            'title' => 'Updated Title',
            'body' => 'Updated body content',
        ]);

        $response->assertStatus(200)
            ->assertJson([
                'title' => 'Updated Title',
                'body' => 'Updated body content',
            ]);
    }

    public function test_can_delete_post()
    {
        $user = User::factory()->create();
        $post = Post::factory()->create(['user_id' => $user->id]);
        Passport::actingAs($user);

        $response = $this->deleteJson("/api/v1/posts/{$post->id}");

        $response->assertStatus(204);
        $this->assertDatabaseMissing('posts', ['id' => $post->id]);
    }
}
```

✓ ရှင်းရှင်းချက်များ

1. Passport အကြောင်း
   Laravel Passport သည် OAuth2 server implementation တစ်ခုဖြစ်ပြီး API authentication အတွက် အသုံးပြုပါတယ်။
2. Authentication Flow

- User သည် register/login လုပ်မယ်။
- Server မှ access token ကို return ပြန်ပါမယ်။
- Client သည် အဲဒီ token ကို အသုံးပြုပြီး API requests များကို လုပ်ဆောင်ပါမယ်။

3. Middleware
   - auth:api middleware သည် request များကို authenticate လုပ်ပါမယ်။
   - Invalid token ဖြစ်ပါက 401 Unauthorized response ကို return ပြန်ပါမယ်။
4. Authorization
   - Post resource များကို ဖန်တီးသူ user သာ update/delete လုပ်နိုင်ပါမယ်။
   - မှာ PostController တွင် authorization check များ ပြုလုပ်ထားပါတယ်။
5. Testing
   - Passport::actingAs() method ကို အသုံးပြုပြီး authenticated user အား simulate လုပ်ပါမယ်။
   - မှန်ကန်တဲ့ responses များရရှိမရှိကို assert လုပ်ပါမယ်။

Laravel Passport ကို အသုံးပြုပြီးတော့ secure API တစ်ခုကို မည်သို့တည်ဆောက်ရမယ် ဆိုတာကို ရှင်းပြသွားတာဖြစ်ပါတယ်။ CRUD operations များနှင့် authentication flow များကို နားလည်သဘောပေါက်ပါက သင့် application အတွက် လိုအပ်တဲ့ API endpoints များကို ထပ်မံဖြည့်စွက်နိုင်ပါတယ်။

Livewire

Livewire

Laravel Livewire သည် ခေတ်သစ် Full-stack Development Framework တစ်ခုဖြစ်ပြီး၊ PHP နဲ့ JavaScript ကို ရောနှောအသုံးပြု၍ Dynamic Web Apps များ တည်ဆောက်ကြတာ ဖြစ်ပါ တယ်။

---

1. Livewire ဆိုတာ ဘာလဲ?

Livewire သည် Laravel အတွက် Frontend Framework တစ်ခုဖြစ်ပြီး အဓိကအားဖြင့်-

- ✅ JavaScript အနည်းဆုံးနဲ့ Dynamic UI များ ဖန်တီးနိုင်ခြင်း
- ✅ PHP နဲ့ပဲ Frontend Logic များ ရေးသားနိုင်ခြင်း
- ✅ AJAX Requests ကို Auto-Handle လုပ်ပေးခြင်း
- ✅ အထူးသဖြင့် SPA (Single Page Apps) များ အတွက် သင့်တော်ခြင်း

Livewire vs Vue.js/React

Feature	Livewire	Vue.js/React
Language	PHP	JavaScript
Rendering	Server-Side	Client-Side
Complexity	Easy	Moderate/Hard
Use Case	CRUD Apps, Dashboards	Complex SPAs

---

2. Livewire ရဲ့အလုပ်လုပ်ပုံ

Livewire သည် အောက်ပါအတိုင်း အဆင့်ဆင့် အလုပ်လုပ်ပါတယ်:

1. Component တစ်ခု ဖန်တီးပါ (PHP Class + Blade View)

2. User Interaction ဖြစ်ပါက (ဥပမာ - Button Click)
3. Livewire က AJAX Request ကို Server သို့ပို့ပါ
4. Server မှ PHP Logic ကို Run ပါ
5. UI ကို Auto-Update လုပ်ပါ (DOM Manipulation လိုမလို)

---

3. Livewire ကို ဘယ်လိုစသုံးမလဲ?

⇨ Installation

Command:

```
composer require livewire/livewire
```

⇨ Component ဖန်တီးခြင်း

Command:

```
php artisan make:livewire Counter
```

ဒီ Command က **အောက်ပါဖိုင်များကို ဖန်တီးပေးပါမယ်**:

- app/Http/Livewire/Counter.php (PHP Logic)
- resources/views/livewire/counter.blade.php (View)

⇨ Example: Simple Counter

Counter.php

Code:

```
<?php
namespace App\Http\Livewire;
use Livewire\Component;

class Counter extends Component
{
    public $count = 0;

    public function increment()
    {
        $this->count++;
```

```
    }

    public function render()
    {
        return view('livewire.counter');
    }
}
```

counter.blade.php

Code:

```
<div>
    <button wire:click="increment">+</button>
    <span>{{ $count }}</span>
</div>
```

အသုံးပြုရန် Blade File ထဲတွင်

Code:

```
<livewire:counter />
```

---

4. Livewire ရဲ့အဓိက Features များ

⇨ Reactive Data Binding

Code:

```
public $name = "John";
```

Code:

```
<input type="text" wire:model="name">
<p>Hello, {{ $name }}!</p> <!-- Auto-update when typing -->
```

⇨Form Handling

Code:

```
public $title, $content;
```

```
public function savePost()
{
    Post::create([
        'title' => $this->title,
        'content' => $this->content,
    ]);
    session()->flash('message', 'Post created!');
}
```

Code:

```
<form wire:submit.prevent="savePost">
    <input wire:model="title" type="text">
    <textarea wire:model="content"></textarea>
    <button type="submit">Save</button>
</form>
```

⇨ Event Handling

Code:

```
public function alert()
{
    $this->emit('showAlert', 'Data saved!');
}
```

Code:

```
<script>
    Livewire.on('showAlert', message => {
        alert(message);
    });
</script>
```

⇨ Pagination, Sorting, Searching

Code:

```
public function render()
{
    return view('livewire.user-list', [
        'users' => User::paginate(10),
    ]);
}
```

---

5. Livewire သုံးဖို့ ဘာကြောင့် သင့်တော်သလဲ?

- ✅ JavaScript နည်းနည်းပဲ သုံးပြီး Dynamic Apps လုပ်နိုင်ခြင်း
- ✅ Laravel Ecosystem နဲ့ အဆင်ပြေစွာ အလုပ်လုပ်နိုင်ခြင်း
- ✅ Real-time Updates များအတွက် အဆင်ပြေခြင်း
- ✅ Learning Curve နည်းခြင်း

---

6. Livewire ရဲ့ အားနည်းချက်များ

- ❌ Complex Animations များအတွက် မသင့်တော်
- ❌ အလွန်ကြီးမားသော SPAs များအတွက် React/Vue ထက် နှေးနိုင်ခြင်း
- ❌ WebSockets (Real-time) အတွက် Laravel Echo နဲ့ ချိတ်ဆက်ရန် လိုအပ်ခြင်း

✍ Short Notes:

Laravel Livewire သည် PHP Developers များအတွက် Modern Web Apps များ တည်ဆောက်ရန် အကောင်းဆုံး Tool တစ်ခုဖြစ်ပါတယ်။ CRUD Apps, Admin Panels, Dashboards များအတွက် အထူးသင့်တော်ပြီး၊ JavaScript ကို နည်းနည်းသုံးပြီး မြန်မြန်ဆန်ဆန် Develop လုပ်နိုင်ပါတယ်။

💻 လေ့ကျင့်ရန်

Laravel Livewire Admin Dashboard Application

✓ Project Structure

```
app/
├── Http/
│   ├── Livewire/
│   │   ├── Users/
│   │   │   ├── UserIndex.php
│   │   │   ├── UserCreate.php
│   │   │   └── UserEdit.php
│   │   ├── Products/
│   │   │   ├── ProductIndex.php
│   │   │   ├── ProductCreate.php
│   │   │   └── ProductEdit.php
│   │   └── Orders/
│   │       ├── OrderIndex.php
│   │       └── OrderShow.php
├── Models/
│   ├── User.php
│   ├── Product.php
│   └── Order.php
database/
├── migrations/
│   ├── create_users_table.php
│   ├── create_products_table.php
│   └── create_orders_table.php
├── seeders/
│   └── DatabaseSeeder.php
resources/
├── views/
│   ├── livewire/
│   │   ├── users/
│   │   │   ├── index.blade.php
```

```
|    |    |    ├──── create.blade.php
|    |    |    └──── edit.blade.php
|    |    ├──── products/
|    |    └──── orders/
|    └──── layouts/
|         └──── admin.blade.php
routes/
├──── web.php
```

✓ Database Setup

⇨ Users Migration

Code:

```
Schema::create('users', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->string('email')->unique();
    $table->string('password');
    $table->enum('role', ['admin', 'user'])->default('user');
    $table->rememberToken();
    $table->timestamps();
});
```

⇨ Products Migration

Code:

```
Schema::create('products', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->text('description');
    $table->decimal('price', 8, 2);
    $table->integer('stock');
    $table->string('image')->nullable();
    $table->timestamps();
});
```

⇨ Orders Migration

Code:

```
Schema::create('orders', function (Blueprint $table) {
    $table->id();
    $table->foreignId('user_id')->constrained();
    $table->decimal('total_amount', 10, 2);
    $table->string('status')->default('pending');
    $table->timestamps();
});
```

```
Schema::create('order_items', function (Blueprint $table) {
    $table->id();
    $table->foreignId('order_id')->constrained();
    $table->foreignId('product_id')->constrained();
    $table->integer('quantity');
    $table->decimal('price', 8, 2);
    $table->timestamps();
});
```

☑ Livewire Components

✓ User Management

⇨ UserIndex Component

Code:

```
namespace App\Http\Livewire\Users;

use Livewire\Component;
use App\Models\User;
use Livewire\WithPagination;

class UserIndex extends Component
{
    use WithPagination;

    public $search = '';
    public $perPage = 10;

    public function render()
```

```
    {
        return view('livewire.users.index', [
            'users' => User::where('name', 'like', '%'.$this->search.'%')
                        ->orWhere('email', 'like', '%'.$this->search.'%')
                        ->paginate($this->perPage)
        ]);
    }

    public function deleteUser($id)
    {
        User::find($id)->delete();
        session()->flash('message', 'User deleted successfully.');
    }
}
```

➪ UserIndex View

Code:

```
<div>
    <div class="flex justify-between items-center mb-4">
        <h2 class="text-2xl font-bold">User Management</h2>
        <a href="{{ route('users.create') }}" class="btn btn-primary">
            Add New User
        </a>
    </div>

    <div class="mb-4">
        <input type="text" wire:model="search" placeholder="Search users...
" class="input input-bordered">
    </div>

    <table class="table w-full">
        <thead>
            <tr>
                <th>ID</th>
                <th>Name</th>
                <th>Email</th>
                <th>Role</th>
                <th>Actions</th>
            </tr>
        </thead>
        <tbody>
            @foreach($users as $user)
                <tr>
```

```
                    <td>{{ $user->id }}</td>
                    <td>{{ $user->name }}</td>
                    <td>{{ $user->email }}</td>
                    <td>{{ ucfirst($user->role) }}</td>
                    <td>
                        <a href="{{ route('users.edit', $user->id) }}" clas
s="btn btn-sm btn-info">Edit</a>
                        <button wire:click="deleteUser({{ $user->id }})" cl
ass="btn btn-sm btn-danger">Delete</button>
                    </td>
                </tr>
            @endforeach
        </tbody>
    </table>

    {{ $users->links() }}
</div>
```

✓ Product Management

⇨ ProductIndex Component

Code:

```
namespace App\Http\Livewire\Products;

use Livewire\Component;
use App\Models\Product;
use Livewire\WithPagination;
use Livewire\WithFileUploads;

class ProductIndex extends Component
{
    use WithPagination, WithFileUploads;

    public $search = '';
    public $perPage = 10;
    public $confirmingProductDeletion = false;
    public $productIdBeingDeleted;

    public function confirmProductDeletion($productId)
    {
        $this->confirmingProductDeletion = true;
        $this->productIdBeingDeleted = $productId;
    }
```

```
    public function deleteProduct()
    {
        Product::find($this->productIdBeingDeleted)->delete();
        $this->confirmingProductDeletion = false;
        session()->flash('message', 'Product deleted successfully.');
    }

    public function render()
    {
        return view('livewire.products.index', [
            'products' => Product::where('name', 'like', '%'.$this->search.
'%')
                                ->orWhere('description', 'like', '%'.$this-
>search.'%')
                                ->paginate($this->perPage)
        ]);
    }
}
```

✓ Order Management

⇨ OrderIndex Component

Code:

```
namespace App\Http\Livewire\Orders;

use Livewire\Component;
use App\Models\Order;
use Livewire\WithPagination;

class OrderIndex extends Component
{
    use WithPagination;

    public $search = '';
    public $perPage = 10;
    public $statusFilter = '';

    public function render()
    {
        return view('livewire.orders.index', [
            'orders' => Order::when($this->search, function($query) {
                            $query->whereHas('user', function($q) {
```

```
                                $q->where('name', 'like', '%'.$this->search
.'%')
                                  ->orWhere('email', 'like', '%'.$this->sea
rch.'%');
                            });
                        })
                        ->when($this->statusFilter, function($query) {
                            $query->where('status', $this->statusFilter);
                        })
                        ->with('user')
                        ->latest()
                        ->paginate($this->perPage)
        ]);
    }

    public function updateStatus($orderId, $status)
    {
        Order::find($orderId)->update(['status' => $status]);
        session()->flash('message', 'Order status updated successfully.');
    }
}
```

✓ Admin Layout

⇨ admin.blade.php

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin Dashboard</title>
    @livewireStyles
    <link href="{{ asset('css/app.css') }}" rel="stylesheet">
</head>
<body class="bg-gray-100">
    <div class="flex h-screen">
        <!-- Sidebar -->
        <div class="w-64 bg-gray-800 text-white">
            <div class="p-4">
                <h1 class="text-2xl font-bold">Admin Panel</h1>
            </div>
            <nav class="mt-6">
```

```
                <x-nav-link href="{{ route('dashboard') }}" :active="reques
t()->routeIs('dashboard')">
                    Dashboard
                </x-nav-link>
                <x-nav-link href="{{ route('users.index') }}" :active="requ
est()->routeIs('users.*')">
                    User Management
                </x-nav-link>
                <x-nav-link href="{{ route('products.index') }}" :active="r
equest()->routeIs('products.*')">
                    Product Management
                </x-nav-link>
                <x-nav-link href="{{ route('orders.index') }}" :active="req
uest()->routeIs('orders.*')">
                    Order Management
                </x-nav-link>
            </nav>
        </div>

        <!-- Main Content -->
        <div class="flex-1 overflow-auto">
            <header class="bg-white shadow">
                <div class="px-4 py-4 flex justify-between items-center">
                    <h2 class="text-xl font-semibold">@yield('title')</h2>
                    <div class="flex items-center space-x-4">
                        <span>{{ Auth::user()->name }}</span>
                        <form method="POST" action="{{ route('logout') }}">
                            @csrf
                            <button type="submit" class="text-sm text-gray-
600 hover:text-gray-900">Logout</button>
                        </form>
                    </div>
                </div>
            </header>

            <main class="p-6">
                @if (session('message'))
                    <div class="alert alert-success mb-4">
                        {{ session('message') }}
                    </div>
                @endif
                @yield('content')
            </main>
        </div>
    </div>
```

```
    @livewireScripts
    <script src="{{ asset('js/app.js') }}"></script>
</body>
</html>
```

✓ Routes

⇨ web.php

Code:

```
Route::middleware(['auth', 'admin'])->group(function () {
    Route::get('/dashboard', function () {
        return view('dashboard');
    })->name('dashboard');

    // User Management
    Route::get('/users', \App\Http\Livewire\Users\UserIndex::class)->name('
users.index');
    Route::get('/users/create', \App\Http\Livewire\Users\UserCreate::class)
->name('users.create');
    Route::get('/users/{user}/edit', \App\Http\Livewire\Users\UserEdit::cla
ss)->name('users.edit');

    // Product Management
    Route::get('/products', \App\Http\Livewire\Products\ProductIndex::class
)->name('products.index');
    Route::get('/products/create', \App\Http\Livewire\Products\ProductCreat
e::class)->name('products.create');
    Route::get('/products/{product}/edit', \App\Http\Livewire\Products\Prod
uctEdit::class)->name('products.edit');

    // Order Management
    Route::get('/orders', \App\Http\Livewire\Orders\OrderIndex::class)->nam
e('orders.index');
    Route::get('/orders/{order}', \App\Http\Livewire\Orders\OrderShow::clas
s)->name('orders.show');
});
```

✓ Testing Results

⇨ User Management Tests

1. User Listing
    - Pagination works correctly
    - Search functionality filters users by name and email
    - Role-based display shows correct user types
2. User Creation
    - Form validation works (required fields, email format)
    - Password is properly hashed
    - Success message appears after creation
3. User Editing
    - Pre-fills form with existing data
    - Updates user information correctly
    - Shows validation errors for invalid inputs

⇨ Product Management Tests

1. Product Listing
    - Displays all products with images
    - Search works for product name and description
    - Delete confirmation modal appears before deletion
2. Product Creation
    - Image upload works (stores in storage and displays)
    - Price validation accepts only numbers
    - Stock quantity must be integer

⇨ Order Management Tests

1. Order Listing
    - Filters orders by status (pending, processing, completed)
    - Shows customer name and total amount
    - Clicking order shows detailed view
2. Order Status Update
    - Admin can change order status

- Status changes reflect immediately
- Confirmation appears after update

✓ Additional Features

⇨ Real-time Notifications

Code:

```
// In OrderStatusUpdated event
public function broadcastOn()
{
    return new PrivateChannel('orders.'.$this->order->id);
}

// In Livewire component
protected $listeners = ['echo-private:orders.{orderId},OrderStatusUpdated'
=> 'refreshOrder'];
```

⇨ Export Functionality

Code:

```
public function exportUsers()
{
    return Excel::download(new UsersExport, 'users.xlsx');
}
```

✍ Shote Notes:

ဒီ Livewire Admin Dashboard တွင် User, Product, Order Management စနစ်များကို အပြည့်အဝ တည်ဆောက်ထားပါတယ်။ Livewire ၏ အဓိက အားသာချက်များဖြစ်သော Reactivity, Form Handling, Pagination များကို အသုံးပြုထားပြီး၊ Clean Code Architecture ဖြင့် ရေးသားထားပါတယ်။

အဆင့်မြင့်ထပ်မံဖြည့်စွက်နိုင်သော Features များ-

- Role-based Permissions (Spatie Laravel-Permission)
- Data Export/Import
- Dashboard Analytics Charts
- Multi-language Support

Git & Github

Git & Github

Git ဆိုတာက version control system တစ်မျိုးဖြစ်ပြီး ကုဒ်တွေကို စနစ်တကျ manage လုပ်ဖို့ အဖွဲ့လိုက်အလုပ်လုပ်ဖို့ အတွက် အသုံးပြုပါတယ်။ Git ကို Linus Torvalds က 2005 ခုနှစ်မှာ Linux kernel development အတွက် ဖန်တီးခဲ့တာဖြစ်ပါတယ်။

GitHub ဆိုတာက Git repositories တွေကို host လုပ်ပေးတဲ့ cloud-based platform တစ်ခုဖြစ်ပါတယ်။ Git commands တွေနဲ့ ကိုယ့်ရဲ့ project တွေကို GitHub ပေါ်မှာ store လုပ်ထားနိုင်ပြီး အခြားသူတွေနဲ့ collaborate လုပ်နိုင်ပါတယ်။

✓ Git Basic Commands

⇨ Git Setup

Command:

```
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

⇨ New Repository စတင်ခြင်း

Command:

```
git init
```

ဒီ command က project folder ထဲမှာ `.git` folder တစ်ခုကို ဖန်တီးပေးပါတယ်။

⇨ File တွေကို Track လုပ်ခြင်း

Command:

```
git add filename.txt  # specific file တစ်ခုကို add လုပ်ခြင်း
git add .             # changed files အားလုံးကို add လုပ်ခြင်း
```

⇨ Changes တွေကို Commit လုပ်ခြင်း

Command:

```
git commit -m "Commit message"
```

⇨ Repository status ကြည့်ခြင်း

Command:

```
git status
```

⇨ Commit history ကြည့်ခြင်း

Command:

```
git log
```

⇨ Branch နဲ့အလုပ်လုပ်ခြင်း

Command:

```
git branch                      # branch list ကြည့်ခြင်း
git branch branch-name          # new branch ဖန်တီးခြင်း
git checkout branch-name        # branch ပြောင်းခြင်း
git merge branch-name           # branch ကို merge လုပ်ခြင်း
```

✓ GitHub အသုံးပြုနည်း

1. New Repository ဖန်တီးခြင်း

1. GitHub website ကိုသွားပါ
2. "+" icon ကိုနှိပ်ပြီး "New repository" ကိုရွေးပါ
3. Repository name ထည့်ပါ
4. Description (optional) ထည့်ပါ
5. Public/Private ရွေးပါ
6. "Create repository" ကိုနှိပ်ပါ

✓ Local Repository ကို GitHub ပေါ်တင်ခြင်း

Command:

```
git remote add origin https://github.com/username/repository-name.git
git branch -M main
git push -u origin main
```

✓ Existing Repository ကို Clone လုပ်ခြင်း

Command:

```
git clone https://github.com/username/repository-name.git
```

✓ Git Push & Pull Requests

⇨ Git Push

Local changes တွေကို remote repository ဆီပို့ဖို့အတွက်:

Command:

```
git push origin branch-name
```

⇨ Git Pull

Remote repository က changes တွေကို local ကိုယူဖို့အတွက်:

Command:

```
git pull origin branch-name
```

✓ Pull Request ဖန်တီးခြင်း (GitHub)

1. GitHub repository ကိုသွားပါ
2. "Pull requests" tab ကိုနှိပ်ပါ
3. "New pull request" button ကိုနှိပ်ပါ
4. Base branch (merge လုပ်မယ့် branch) နဲ့ compare branch (သင့်ရဲ့ changes တွေပါတဲ့ branch) ကိုရွေးပါ
5. Changes တွေကိုစစ်ဆေးပါ
6. "Create pull request" button ကိုနှိပ်ပါ
7. Title နဲ့ description ရေးပါ
8. "Create pull request" ကိုနှိပ်ပါ

✓ Collaboration Workflow

⇨ Main repository ကို fork လုပ်ပါ (GitHub ပေါ်မှာ)

⇨ ကိုယ့် computer local မှာ clone လုပ်ပါ

Command:

```
git clone https://github.com/your-username/repository.git
```

⇨ New branch တစ်ခုဖန်တီးပါ

Command:

```
git checkout -b feature-branch
```

⇨ Changes တွေလုပ်ပါ၊ add နဲ့ commit လုပ်ပါ

Command:

```
git add .
git commit -m "Your commit message"
```

⇨ Changes တွေကို push လုပ်ပါ

Command:

```
git push origin feature-branch
```

⇨ GitHub ပေါ်မှာ pull request ဖန်တီးပါ

⇨ Project maintainer က review လုပ်ပြီး merge လုပ်မယ်

✓ Git Concepts များ

1. Working Directory: သင်အလုပ်လုပ်နေတဲ့ files တွေရှိတဲ့နေရာ
2. Staging Area: commit လုပ်ဖို့အဆင်သင့်ဖြစ်နေတဲ့ changes တွေ
3. Repository: commit လုပ်ပြီးသား changes တွေသိမ်းထားတဲ့နေရာ
4. Branch: project ရဲ့ parallel version တစ်ခု
5. Merge: branch တစ်ခုရဲ့ changes တွေကို အခြား branch တစ်ခုထဲပေါင်းထည့်ခြင်း
6. Conflict: အတူတူ file တွေကို နှစ်ယောက်က တစ်ချိန်တည်း modify လုပ်မိရင် ဖြစ်တတ်တဲ့အရာ

✓ အကြံပြုချက်များ

1. Commit messages တွေကို ရှင်းရှင်းလင်းလင်းရေးပါ
2. မကြာခဏ commit လုပ်ပါ (small changes တွေနဲ့)

3. Feature တစ်ခုချင်းစီအတွက် separate branch တွေသုံးပါ
4. Pull request မလုပ်ခင် changes တွေကို ကိုယ်တိုင်ပြန်စစ်ပါ
5. .gitignore file ကိုသုံးပြီး unnecessary files တွေကို track မလုပ်မိအောင်လုပ်ပါ

Git နဲ့ GitHub ကို ပုံမှန်အသုံးပြုရင်းနဲ့ နားလည်မှုတိုးလာမယ်ဆိုတာ သေချာပါတယ်။ အစပိုင်းမှာ နည်းနည်းရှုပ်ထွေးနိုင်ပေမယ့် လေ့ကျင့်သင်ယူဖို့ အချိန်ပေးပါက version control ရဲ့ အကျိုးကျေးဇူးတွေကို သိသိသာသာ ခံစားရမှာဖြစ်ပါတယ်။

👉 လေ့ကျင့်ရန်:

✓ Git Collaboration Workflow for Web App Development

ဒီ git collaboration workflow မှာ Mg Mg (Chief Web Developer), Su Su နဲ့ Aung Aung တို့ 3 ယောက်က simple web app တစ်ခုကို collaborate လုပ်ပြီးရေးသားကြမယ့် online web application development အဆင့်ဆင့်ကို git commands တွေနဲ့အတူ ရှင်းပြပါမယ်။

✓ Mg Mg ရဲ့အစပြုလုပ်ဆောင်ချက်များ

⇨ Local Repository စတင်ခြင်း

Command:

```
# Project folder ဖန်တီးပါ
mkdir web_app1
cd web_app1

# Git repository စတင်ပါ
git init

# Home page file ဖန်တီးပါ
echo "<h1>Welcome to Home Page</h1>" > index.html
```

```
# Initial commit လုပ်ပါ
git add index.html
git commit -m "Initial commit with home page"
```

⇨ GitHub Repository ဖန်တီးခြင်း

1. GitHub ကိုသွားပါ
2. "New repository" ကိုနှိပ်ပါ
3. Repository name ကို "web_app1" လို့ပေးပါ
4. "Create repository" ကိုနှိပ်ပါ

⇨Local Repository ကို GitHub ပေါ်တင်ခြင်း

Command:

```
# Remote repository ကို add လုပ်ပါ
git remote add origin https://github.com/MgMg/web_app1.git

# Main branch ကို push လုပ်ပါ
git push -u origin main
```

✓ Su Su ရဲ့အလုပ်လုပ်ပုံ

⇨ Repository ကို Clone လုပ်ခြင်း

Code:

```
# Mg Mg ရဲ့repository ကို clone လုပ်ပါ
git clone https://github.com/MgMg/web_app1.git
cd web_app1
```

⇨ New Branch ဖန်တီးပြီး About Page ရေးသားခြင်း

Code:

```
# New branch ဖန်တီးပါ
git checkout -b su-su-about-page

# About page ဖန်တီးပါ
echo "<h1>About Us</h1><p>This is our about page</p>" > about.html
```

```
# Changes တွေကို commit လုပ်ပါ
git add about.html
git commit -m "Add about page by Su Su"
```

⇨ Changes တွေကို GitHub ပေါ်တင်ခြင်း

Command:

```
# Branch ကို push လုပ်ပါ
git push origin su-su-about-page
```

⇨ Pull Request ဖန်တီးခြင်း

1. GitHub repository ကိုသွားပါ
2. "Pull requests" tab ကိုနှိပ်ပါ
3. "New pull request" ကိုနှိပ်ပါ
4. "su-su-about-page" branch ကိုရွေးပါ
5. "Create pull request" ကိုနှိပ်ပါ
6. Title နဲ့ description ရေးပါ (ဥပမာ - "Add about page")
7. "Create pull request" ကိုနှိပ်ပါ

✓ Mg Mg က Pull Request Review နဲ့ Merge လုပ်ခြင်း

1. GitHub ပေါ်က pull request ကိုသွားပါ
2. Changes တွေကိုစစ်ဆေးပါ
3. အဆင်ပြေရင် "Merge pull request" ကိုနှိပ်ပါ
4. "Confirm merge" ကိုနှိပ်ပါ
5. (Optional) Branch ကို delete လုပ်နိုင်ပါတယ်

✓ Aung Aung ရဲ့အလုပ်လုပ်ပုံ

⇨ Latest Version ကို Clone လုပ်ခြင်း

Command:

```
# Latest repository ကို clone လုပ်ပါ
```

```
git clone https://github.com/MgMg/web_app1.git
cd web_app1
```

⇨ New Branch ဖန်တီးပြီး Contact Page ရေးသားခြင်း

Command:

```
# New branch ဖန်တီးပါ
git checkout -b aung-aung-contact-page

# Contact page ဖန်တီးပါ
echo "<h1>Contact Us</h1><p>Email: contact@example.com</p>" > contact.html

# Changes တွေကို commit လုပ်ပါ
git add contact.html
git commit -m "Add contact page by Aung Aung"
```

⇨ Changes တွေကို GitHub ပေါ်တင်ခြင်း

Command:

```
# Branch ကို push လုပ်ပါ
git push origin aung-aung-contact-page
```

✓ Pull Request ဖန်တီးခြင်း

1. GitHub repository ကိုသွားပါ
2. "Pull requests" tab ကိုနှိပ်ပါ
3. "New pull request" ကိုနှိပ်ပါ
4. "aung-aung-contact-page" branch ကိုရွေးပါ
5. "Create pull request" ကိုနှိပ်ပါ
6. Title နဲ့ description ရေးပါ (ဥပမာ - "Add contact page")
7. "Create pull request" ကိုနှိပ်ပါ

✓ Mg Mg က နောက်ဆုံး Pull Request ကို Review နဲ့ Merge လုပ်ခြင်း

1. GitHub ပေါ်က pull request ကိုသွားပါ
2. Changes တွေကိုစစ်ဆေးပါ
3. အဆင်ပြေရင် "Merge pull request" ကိုနှိပ်ပါ

4. "Confirm merge" ကိုနှိပ်ပါ

✓အားလုံးက Latest Version ကို Sync လုပ်ခြင်း

⇨ Su Su ရဲ့ Local Repository Update

Command:

```
cd web_app1  # Su Su's local repository
git checkout main
git pull origin main
```

⇨ Aung Aung ရဲ့ Local Repository Update

Command:

```
cd web_app1  # Aung Aung's local repository
git checkout main
git pull origin main
```

✓ Security Best Practices

⇨ **Authentication**:

Command:

```
# SSH key သုံးပြီး secure connection လုပ်ပါ

git remote set-url origin git@github.com:MgMg/web_app1.git
```

⇨ **Branch Protection**:

- Mg Mg က GitHub repository settings မှာ "main" branch ကို protect လုပ်ထားပါ
- Direct push ကို ပိတ်ထားပြီး pull request ကနေသာ merge လုပ်ရမယ့် စနစ်ချမှတ်ပါ

⇨ **Code Review**:

- Pull request တိုင်းကို Mg Mg (Chief Developer) က approve လုပ်မှသာ merge လုပ်ရမယ်

⇨ .gitignore ဖိုင်သုံးပါ:

Command:

```
# Node modules, environment variables တွေကို ignore လုပ်ပါ
```

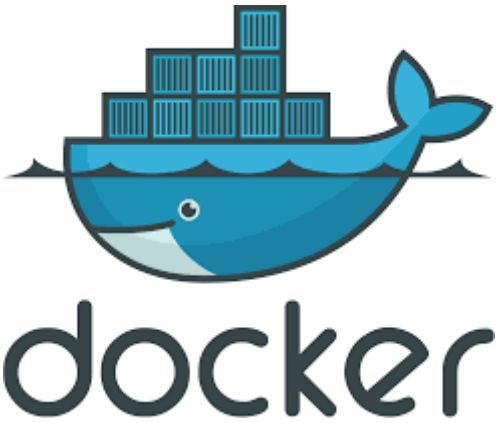
```
echo "node_modules\n.env" > .gitignore
git add .gitignore
git commit -m "Add .gitignore file"
git push origin main
```

✓ Workflow Summary

1. Mg Mg က main repository ကိုစတင်ပါတယ်
2. Developer တစ်ယောက်ချင်းစီက:
    - Repository ကို clone လုပ်ပါတယ်
    - Feature branch တစ်ခုဖန်တီးပါတယ်
    - သူတို့ရဲ့ changes တွေကို commit လုပ်ပါတယ်
    - Branch ကို push လုပ်ပါတယ်
    - Pull request ဖန်တီးပါတယ်
3. Mg Mg က changes တွေကို review လုပ်ပြီး merge လုပ်ပါတယ်
4. အားလုံးက main branch ကို update လုပ်ပါတယ်

ဒီ workflow ကို အသုံးပြုခြင်းဖြင့် web app development လုပ်ငန်းစဉ်တွေကို စနစ်တကျနဲ့ secure ဖြစ်စွာ manage လုပ်နိုင်မှာဖြစ်ပါတယ်။

Docker



Docker

Docker သည် containerization technology တစ်ခုဖြစ်ပြီး application တွေကို isolated environment များမှာ run နိုင်အောင် ကူညီပေးပါတယ်။ Laravel နှင့် Docker ကို အသုံးပြုခြင်းအားဖြင့် development process ကို ပိုမိုလွယ်ကူမြန်ဆန်စွာ ဆောင်ရွက်နိုင်မှာ ဖြစ်ပါတယ်။

Laravel အတွက် Docker ကို အသုံးပြုခြင်းအားဖြင့် ရရှိမယ့် အကျိုးကျေးဇူးတွေကတော့ -

- Development environment တစ်ခုတည်းကို team member အားလုံးနှင့် share လုပ်နိုင်ခြင်း
- Local machine တွင် dependency ပြဿနာများ မရှိတော့ခြင်း
- Production environment နှင့် ဆင်တူသော environment တွင် develop လုပ်နိုင်ခြင်း

တို့ပဲ ဖြစ်ပါတယ်။

Laravel Docker Setup အခြေခံအဆင့်များ

✓ Dockerfile ဖိုင်ဖန်တီးခြင်း

Code:

```
FROM php:8.2-fpm

# Dependencies တပ်ဆင်ခြင်း
RUN apt-get update && apt-get install -y \
    git \
    curl \
    libpng-dev \
    libonig-dev \
    libxml2-dev \
    zip \
    unzip

# PHP Extensions တပ်ဆင်ခြင်း
```

Command:

```
RUN docker-php-ext-install pdo_mysql mbstring exif pcntl bcmath gd
```

```
# Composer install လုပ်ခြင်း
COPY --from=composer:latest /usr/bin/composer /usr/bin/composer

# Working directory သတ်မှတ်ခြင်း
WORKDIR /var/www

CMD ["php-fpm"]
```

✓ docker-compose.yml ဖိုင်ဖန်တီးခြင်း

Code:

```
version: '3.8'

services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
    volumes:
      - ./:/var/www
    networks:
      - laravel

  webserver:
    image: nginx:alpine
    ports:
      - "8000:80"
    volumes:
      - ./:/var/www
      - ./nginx/default.conf:/etc/nginx/conf.d/default.conf
    depends_on:
      - app
    networks:
      - laravel

  db:
    image: mysql:5.7
    environment:
      MYSQL_DATABASE: laravel
      MYSQL_ROOT_PASSWORD: secret
    ports:
      - "3306:3306"
    volumes:
      - dbdata:/var/lib/mysql
```

```
    networks:
      - laravel

volumes:
  dbdata:

networks:
  laravel:
    driver: bridge
```

✓ Nginx Configuration

⇨ nginx/default.conf ဖိုင်တွင် အောက်ပါအတိုင်း configure လုပ်ပါမည်။

Config:

```
server {
    listen 80;
    index index.php index.html;
    server_name localhost;
    root /var/www/public;

    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    location ~ \.php$ {
        fastcgi_pass app:9000;
        fastcgi_index index.php;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        include fastcgi_params;
    }
}
```

✓ Laravel Docker Project စတင်အသုံးပြုနည်း

1. Docker containers များ start လုပ်ရန်:

Command:

```
docker-compose up -d
```

2. Laravel dependencies တပ်ဆင်ရန်:

   Command:

```
docker-compose exec app composer install
```

3. .env file ကို copy လုပ်ပြီး configure လုပ်ရန်:

   Command:

```
cp .env.example .env
```

4. Application key generate လုပ်ရန်:

   Command:

```
docker-compose exec app php artisan key:generate
```

5. Database migrate လုပ်ရန်:

   Command:

```
docker-compose exec app php artisan migrate
```

✓ အသုံးဝင်သော Docker Commands များ

- Containers များအားလုံးကို stop လုပ်ရန်:

  Command:

```
docker-compose down
```

- Specific service ကို restart လုပ်ရန်:

  Command:

```
docker-compose restart app
```

- Running containers များကို ကြည့်ရန်:

  Command:

```
docker ps
```

- Logs များကို ကြည့်ရန်:

  Command:

```
docker-compose logs
```

လေ့ကျင့်ရန်-

Windows OS ကိုသုံးပြီး Hostinger VPS မှာ Docker ဖြင့် Laravel Web App Deploy လုပ်နည်း

✓ ကြိုတင်ပြင်ဆင်မှုများ

Hostinger ဘက်မှာ-

1. Hostinger VPS ဝယ်ယူထားရန်
2. SSH access ရှိရန် (Root password သို့မဟုတ် SSH key)
3. Domain name ကို VPS နှင့် ချိတ်ဆက်ထားရန်

Windows ဘက်မှာ-

1. MobaXterm (သို့) PuTTY တပ်ဆင်ထားရန်
2. WinSCP ဖိုင်လွှဲအပြန်အလှန်လုပ်ရန်

✓ Hostinger VPS သို့ SSH ချိတ်ဆက်ခြင်း

1. MobaXterm ဖွင့်ပါ
2. New Session > SSH ကိုရွေးပါ
3. Hostinger ပေးထားသော IP address ထည့်ပါ
4. Username: root
5. Password: Hostinger ပေးထားသော password

✓ VPS တွင် Docker တပ်ဆင်ခြင်း

Code:

```
# Docker တပ်ဆင်ခြင်း
sudo apt update
sudo apt install -y apt-transport-https ca-certificates curl software-prope
rties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmo
r -o /usr/share/keyrings/docker-archive-keyring.gpg
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/
docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_
release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/
null
sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io

# Docker Compose တပ်ဆင်ခြင်း
sudo curl -L "https://github.com/docker/compose/releases/download/v2.20.3/d
ocker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose

# Docker ကို root မလိုဘဲ အသုံးပြုနိုင်ရန်
sudo usermod -aG docker $USER
newgrp docker
```

✓ Windows မှ Project ဖိုင်များကို VPS သို့ပို့ခြင်း

1. WinSCP ဖွင့်ပါ
2. Hostinger VPS ၏ IP address နှင့် credentials များထည့်ပါ
3. Local (Windows) ဘက်ရှိ Laravel project folder ကို VPS ၏ /home/username အောက်သို့ upload လုပ်ပါ

✓ Docker Configuration ဖိုင်များ ပြင်ဆင်ခြင်း

⇨ docker-compose.prod.yml ဖိုင်

Code:

```
version: '3.8'

services:
```

```
  app:
    build:
      context: .
      dockerfile: Dockerfile
    restart: always
    volumes:
      - ./:/var/www
    environment:
      - APP_ENV=production
    networks:
      - laravel_network

  webserver:
    image: nginx:alpine
    restart: always
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./:/var/www
      - ./nginx:/etc/nginx/conf.d
    depends_on:
      - app
    networks:
      - laravel_network

  db:
    image: mysql:5.7
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: your_strong_password
      MYSQL_DATABASE: laravel_prod
    volumes:
      - mysql_data:/var/lib/mysql
    networks:
      - laravel_network

volumes:
  mysql_data:

networks:
  laravel_network:
    driver: bridge
```

⇨ Nginx Configuration (nginx/app.conf)

Code:

```
server {
    listen 80;
    server_name yourdomain.com;
    root /var/www/public;

    index index.php;

    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    location ~ \.php$ {
        fastcgi_pass app:9000;
        fastcgi_param SCRIPT_FILENAME $realpath_root$fastcgi_script_name;
        include fastcgi_params;
    }

    location ~ /\.(?!well-known).* {
        deny all;
    }
}
```

✓ Laravel App Configuration

Command:

```
cd /home/username/your-project-folder
cp .env.example .env
nano .env
```

.env ဖိုင်တွင် အောက်ပါအတိုင်း ပြင်ဆင်ပါ

Config:

```
APP_ENV=production
APP_DEBUG=false
APP_URL=http://yourdomain.com

DB_CONNECTION=mysql
DB_HOST=db
DB_PORT=3306
DB_DATABASE=laravel_prod
DB_USERNAME=root
DB_PASSWORD=your_strong_password
```

✓ Docker Containers များ စတင်ခြင်း

Command:

```
docker-compose -f docker-compose.prod.yml build
docker-compose -f docker-compose.prod.yml up -d
```

✓ Laravel Setup များ ပြုလုပ်ခြင်း

Command:

```
docker-compose -f docker-compose.prod.yml exec app composer install --no-de
v
docker-compose -f docker-compose.prod.yml exec app php artisan key:generate
docker-compose -f docker-compose.prod.yml exec app php artisan migrate --fo
rce
docker-compose -f docker-compose.prod.yml exec app php artisan storage:link
docker-compose -f docker-compose.prod.yml exec app php artisan optimize
```

✓ SSL Certificate တပ်ဆင်ခြင်း (Optional)

Command:

```
sudo apt install certbot python3-certbot-nginx
sudo certbot --nginx -d yourdomain.com
```

✓ Hostinger VPS တွင် Firewall Setting များ

1. Hostinger control panel သို့ဝင်ပါ
2. VPS > Firewall section သို့သွားပါ
3. Port 80, 443, 22 များကို allow လုပ်ပါ

✓ Windows မှ Application ကို စစ်ဆေးခြင်း

Web browser တွင် အောက်ပါ address ကိုရိုက်ထည့်ပါ

url:

```
http://yourdomain.com
```

✓ အရေးကြီးသော အချက်များ

1. Hostinger VPS တွင် resource နည်းပါက docker-compose.prod.yml တွင် memory limit သတ်မှတ်ပါ
2. ပုံမှန် backup များပြုလုပ်ပါ
3. .env ဖိုင်တွင် sensitive data များကို မဖော်ပြမိစေရန် သတိထားပါ
4. Hostinger ဘက်တွင် cron job များသတ်မှတ်ရန် လိုအပ်ပါက docker-compose exec command များကိုအသုံးပြုပါ

ဤအဆင့်များအတိုင်း လုပ်ဆောင်ပါက Hostinger VPS ပေါ်တွင် Docker ဖြင့် Laravel application ကို အဆင်ပြေစွာ deploy လုပ်နိုင်မည် ဖြစ်ပါသည်။

Web Deployment

Web Deployment

☑ ကြိုတင်ပြင်ဆင်မှုများ

✓ Hostinger website ဘက်မှာ လုပ်ဆောင်ရမယ့် အချက်များ

1. Hostinger မှ Shared Hosting သို့မဟုတ် VPS ဝယ်ယူထားရန်
2. cPanel သို့မဟုတ် SSH access ရှိရန်
3. Domain name ကို Hostinger account နှင့် ချိတ်ဆက်ထားရန်

✓ Windows ဘက်မှာ လုပ်ဆောင်ရမယ့် အချက်များ

1. FileZilla (FTP client) တပ်ဆင်ထားရန်
2. Visual Studio Code သို့မဟုတ် အခြား code editor

✓ Laravel Project ကို Windows တွင် ပြင်ဆင်ခြင်း

.env ဖိုင် ပြင်ဆင်ခြင်း

Config:

```
APP_ENV=production
APP_DEBUG=false
APP_URL=http://yourdomain.com

DB_CONNECTION=mysql
DB_HOST=hostinger-mysql-host
DB_PORT=3306
DB_DATABASE=your_db_name
DB_USERNAME=your_db_user
DB_PASSWORD=your_db_password
```

✓ Production အတွက် optimize

Command:

```
composer install --optimize-autoloader --no-dev
php artisan config:cache
php artisan route:cache
php artisan view:cache
```

✓ Hostinger cPanel သို့ ဖိုင်များ Upload လုပ်ခြင်း

1. FileZilla ဖွင့်ပါ
2. Hostinger ပေးထားသော FTP credentials များထည့်ပါ
    - Host: yourdomain.com
    - Username: your_cpanel_username
    - Password: your_ftp_password
    - Port: 21
3. Local ဘက်ရှိ Laravel project folder ကို public_html folder အတွင်းသို့ upload လုပ်ပါ
4. အောက်ပါ folder များကို permissions 755 ပေးပါ
    - storage/
    - bootstrap/cache/

✓ Hostinger Database Setup

1. cPanel တွင် MySQL Databases ကိုနှိပ်ပါ
2. New database ဖန်တီးပါ
3. Database user ဖန်တီးပြီး database နှင့် ချိတ်ဆက်ပါ
4. .env ဖိုင်တွင် database information များ update လုပ်ပါ

✓ Hostinger ပေါ်တွင် Laravel အတွက် Configuration

⇨ public_html/.htaccess ဖိုင်

Code:

```
<IfModule mod_rewrite.c>
    <IfModule mod_negotiation.c>
        Options -MultiViews -Indexes
    </IfModule>

    RewriteEngine On

    # Handle Authorization Header
    RewriteCond %{HTTP:Authorization} .
    RewriteRule .* - [E=HTTP_AUTHORIZATION:%{HTTP:Authorization}]

    # Redirect Trailing Slashes If Not A Folder...
    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteCond %{REQUEST_URI} (.+)/$
    RewriteRule ^ %1 [L,R=301]

    # Send Requests To Front Controller...
    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteRule ^ index.php [L]
</IfModule>
```

⇨ public/index.php ဖိုင် ပြင်ဆင်ခြင်း

Code:

```
require __DIR__.'/../vendor/autoload.php';
$app = require_once __DIR__.'/../bootstrap/app.php';
```

✓ Hostinger ပေါ်တွင် PHP Version ပြောင်းခြင်း

1. cPanel တွင် "Select PHP Version" ကိုနှိပ်ပါ
2. PHP 8.0 သို့မဟုတ် သင့် Laravel version နှင့် ကိုက်ညီသော version ကိုရွေးပါ
3. Required extensions များ enable လုပ်ပါ:

- fileinfo
- pdo_mysql
- mbstring
- openssl
- tokenizer

✓ Application ကို စမ်းသပ်ခြင်း

⇨ Web browser တွင် အောက်ပါ address ကိုရိုက်ထည့်ပါ

url:

```
http://yourdomain.com
```

✓ အရေးကြီးသော အချက်များ

1. Security:
   - .env ဖိုင်တွင် APP_KEY ပါရှိရန်
   - storage/ နှင့် bootstrap/cache/ folder များကို proper permissions ပေးရန်
   - debug mode ကို false ထားရန်
2. Performance:
   - Hostinger cache စနစ်များကို အသုံးပြုပါ
   - Cloudflare CDN ချိတ်ဆက်အသုံးပြုပါ
3. Backup:
   - ပုံမှန် database backup များပြုလုပ်ပါ
   - File backup များပြုလုပ်ပါ

How to land a Job

How to land a job

Laravel Developer အဖြစ် အလုပ် ရဖို့ဆိုရင် Technical Skills, Projects, Interview Preparation နဲ့ Job Hunting Strategies တွေကို အဆင့်ဆင့်ပြင်ဆင်ဖို့လိုပါတယ်။

---

1. Laravel ကို အခြေခံကစပြီး ကျွမ်းကျင်အောင် လေ့လာပါ

⇨ Core Laravel Concepts များ

- Routing, Controllers, Views (MVC Pattern)
- Eloquent ORM (Database Relationships, Query Builder)
- Middleware, Authentication (Sanctum, Passport)
- API Development (RESTful, JSON Responses)
- Blade Templating, Livewire (Frontend Integration)

⇨ Advanced Topics များ

- Testing (PHPUnit, Pest)
- Queue & Jobs (Redis, Horizon)
- Events & Listeners
- Package Development
- Performance Optimization (Caching, Eager Loading)

📌 လေ့လာရန် Resources:

- Laravel Documentation
- Laracasts (Video Tutorials)
- Codecourse

2. Portfolio Project တွေ တည်ဆောက်ပါ

အလုပ်ရှင်တွေက လက်တွေ့ Project Experience ကို ကြည့်တတ်တာကြောင့် Real-world Apps တွေ ဆောက်ထားဖို့လိုပါမယ်။ အနည်းဆုံး web project ၅ ခုလောက်တော့ ဆောက်ထားင့်ပါတယ်။

Project Examples:

- ✅ E-commerce Website (Product CRUD, Payment Gateway)
- ✅ Blog with Admin Panel (User Roles, Comments)
- ✅ Task Management System (Drag & Drop, AJAX)
- ✅ API for Mobile App (JWT Authentication)
- ✅ Real-time Chat App (Laravel + Pusher)

📌 GitHub မှာ Public Repo တင်ပါ:

- Clean Code ရေးပါ
- README.md ကောင်းကောင်းရေးပါ
- Live Demo (Heroku, Vercel) ချိတ်ပါ

✓ Job-Ready Skills တွေ ပြင်ဆင်ပါ

⇨ PHP & Database

- OOP, SOLID Principles
- MySQL, PostgreSQL (Query Optimization)
- Redis, Caching Mechanisms

⇨ Frontend Integration

- JavaScript (Vue.js/React if possible)
- Tailwind CSS, Bootstrap
- AJAX, Axios

⇨ DevOps & Deployment

- Docker, Laravel Sail
- CI/CD (GitHub Actions, Envoyer)
- Linux Server Management (Nginx, Apache)

☑ Job Application အတွက် ဘာတွေပြင်ရမလဲ?

⇨ CV/Resume ပြင်ဆင်ခြင်း

- Laravel Projects များကို Highlight
- Skills Section တွင် Frameworks, Tools များ ထည့်ပါ
- GitHub, Portfolio Link ထည့်ပါ

⇨ LinkedIn & Job Portals

- LinkedIn Profile ကို Optimize လုပ်ပါ
- Upwork, Fiverr, RemoteOK တွင် Profile ဖွင့်ပါ
- Local Job Portals (JobNet, MyJobs) တွင် လျှောက်ပါ

⇨ Interview Preparation

✅ Technical Questions:

- Laravel Lifecycle
- Eloquent vs Query Builder
- Service Container
- CSRF Protection

✅ Coding Challenges:

- Small Task (CRUD, API)
- Bug Fixing

✅ Soft Skills:

- Problem-Solving Approach
- Teamwork Experience

✓ Freelancing/Internship ကနေ စပါ

အကယ်၍ Junior Level မှာဆိုရင် Internship သို့မဟုတ် Freelance Gigs များကနေ စပြီး Experience ယူနိုင်ပါတယ်။

📌 Freelance Platforms:

- Upwork
- Fiverr
- PeoplePerHour

📌 Local Internships:

- Tech Companies in Myanmar
- Remote Internships (Global)

✓ Social Networking & Community Join ပါ

- Laravel Myanmar Community (Facebook Groups)
- Meetups, Webinars တက်ပါ
- Open Source Contributions (GitHub)

✍ Short Notes:

Laravel Developer အဖြစ် Job ရဖို့ဆိုရင်:

1. Laravel Core Concepts ကို နားလည်အောင် လေ့လာပါ
2. Portfolio Projects တွေ တည်ဆောက်ပါ
3. GitHub, LinkedIn ကို Professional ဖြစ်အောင်လုပ်ပါ
4. Job Applications များများလျှောက်ပါ
5. Interview အတွက် Technical & Soft Skills ပြင်ဆင်ပါ

အချိန်ပေးပြီး လေ့လာမယ်ဆိုရင် 3-6 လ အတွင်း Laravel Job ကောင်းကောင်းတစ်ခု ရနိုင်ပါတယ်။

"Keep Coding, Keep Applying!"

■ References

1. https://laravel.com
2. Shawn McCool. (2013). Laravel Starter
3. Chuck Heintzelman. (2016). Laravel 5.x Beauty
4. Martin Bean. (2015). Laravel 5 Essentials
5. Paul Redmond. (2022). Laravel API Development
6. Brent Roose. (2022). Laravel Beyond CRUD
7. https://www.tutorialspoint.com/laravel
8. https://openai.com/index/chatgpt
9. https://chat.deepseek.com

■ စားရေးသူ၏ ကိုယ်ရေးအကျဉ်း

ဤစာအုပ်ကိုရေးသားပြုစုသူကတော့ ကျွန်တော် ဆရာတင်မိုင်ဇော် ဖြစ်ပါတယ်။ ကျွန်တော်က မြစ်ကြီးနားမြို့အထက (၁) မှာ အထက်တန်းကို ၁၉၉၄ မှာ အောင်မြင်ခဲ့ပါတယ်။ ပြီးတော့ ကျွန်တော် ဖိလိပိုင်နိုင်ငံ University of the Philippines (Los Banos) မှာ B.Sc Computer Science ကိုဆက်လက်ပညာသင်ယူခဲ့ပါတယ်။ ၂၀၀၄ မှာ B.Sc Computer Science နဲ့ ဘွဲ့ရကျောင်းပြီးခဲ့ပါတယ်။ ၂၀၀၅ မှာ မြန်မာပြည်ပြန်လာပြီး မြစ်ကြီးနား ဇာတိမြို့မှာ Northern City Computer Training Center ကွန်ပျူတာသင်တန်းကျောင်းကို စတင်တည်ထောင်ဖွင့် လှစ်ခဲ့ပါတယ်။ လေ့လာဆည်းပူးခဲ့တဲ့ programming IT ဘာသာရပ်တွေကို သင်ကြားပို့ချခဲ့တာ ဖြစ်ပါတယ်။ ၂၀၀၉ မှာ မလေးရှားနိုင်ငံ Kualalumpur မှာ Zepto IT Solution လို့ခေါ်တဲ့ အိုင်တီ Company မှာ Software Developer (programmer) အဖြစ် ၃ နှစ်တာ အလုပ်လုပ်ခဲ့ပါတယ်။ ကျန်းမာရေးအခြေနေ ကြောင့် ရန်ကုန်မြို့ကို ပြန်လာပြီး ဆေးကုသရင်း ရန်ကုန်မြို့မှာပဲ ValueStar Computer Institute မှာ ၆ နှစ်တာ IT Lecturer အနေနဲ့ရော IT Department Head အနေနဲ့ရော ဆက်လက်ခြေချခဲ့ပါတယ်။ ၂၀၁၇ မှာ ကိုယ်ပိုင် အိုင်တီသင်တန်းကျောင်းအဖြစ် Northern City Center နာမည်နဲ့ အရင်က မြစ်ကြီးနားမှာ တည်ထောင်ခဲ့ဖူး တဲ့ နာမည်ကို ပြန်လည်အသုံးပြုကာ ဖွင့်လှစ်ထားပြီး ၂၀၂၅ လက်ရှိအချိန်အထိ သင်တန်းကျောင်းကို ဦးစီးလုပ်ကိုင်နေဆဲ ဖြစ်ပါတယ်။

■ စာရေးသူ၏ အိုင်တီအတွေ့အကြုံ မှတ်တမ်းများ

✓ Popular Web projects –

- Japan Used Car Sales & Show room
  https://www.sbtjapan.com/sbt-myanmar/
- Myanmar Traditional Boxing
  https://www.myanmartraditionalboxing.com.mm
- Myanmar Agriculture Machinery & Products
  https://tptyeeshinn.com.mm
- Real Estate
  https://www.saikhungnoung.com
- Malaysia Minimart
  https://familystore.com.my
- China Products & Fashion Sales
  https://youhome.space/
- Online Payment System
  https://ucapay.com.mm

✓ Popular Point of Sales Application Projects –

- Malaysia Family Store Desktop Application and Mobile Application
- Myitkyina iGu Café & Gallery Desktop Application
- Myitkyina Hospital Blood Bank Desktop Application
- Myitkyina Fuji Food & Drinks Desktop Application
- Yangon Joyzone Stock Controller Desktop Application and Mobile Application
- Yangon Malihka Restaurant Desktop Application and Mobile Application
- Yangon Yuri Café Mobile Application and Mobile Application
- Yangon Gracevines Agarwood Desktop Application and Mobile Application

■ စာရေးသူ၏ Documentary Photos

ပျော်ရွှင် ချမ်းမြေ့ကြပါစေ။

Northern City